F-PACK

F - package for input/output
(Version 0.86/00)
(C) V.B. 1990

DRAFT
11. October 1993

FPACK is a general stand-alone package of FORTRAN77 and C programs for the machine-independent input/output of data blocks. A word-format conversion between local and other word-formats is performed using a format description of the data blocks, if data are exchanged between different computers. Exchange of data is possible via networks and tapes, cassettes etc. F-records contain a record key for fast access to a subset of the data. Unformatted and formatted access is supported, in addition keyed access on certain machines. At present only 32-bit computers are supported.


```
In case of any problems please contact Pavel Binko (R00PAV@DHHDESY3)


Authors of the system:

   Volker Blobel       (II. Institut fuer Experimentalphysik,
                            University of Hamburg, Germany)

   Pavel Binko         (Institute of Scientific Instruments of the
                            Academy of Sciences, Brno, CSFR)

   Zbigniew Szkutnik   (Institute of Mathematics, University
                            of Mining and Metallurgy Cracow, Poland)
```

(This document was typeset in LaTeXfrom the plain-text version by Timothy Paul Smith
University of New Hampshire - May 1995)

Information on DESY IBM/MVS:

Load library: HERA01.H1.FPACK.LOAD

The BOS interface routines are contained in the H1 Utility library, module steering subroutines are now called internally. The following H1 libraries (XA) are necessary for BOS + FPACK applications:

```
HERA01.H1UTIL.LOAD
HERA01.H1.BOS.LOAD
HERA01.H1.LOOK.LOAD
R01UTL.CERN.PRO.PACKLIB
```

The compilation of user programs containing the BOS common BCS requires (because of XA) the specification of a special compilation parameter in the EXEC VFORT... statement:

```
CPRM='DC(BCS)'
```

Contents

```
General information on the CMZ library
-------------------------------------


Load library      : HERA01.H1.FPACK.LOAD          version 0.86/00
                    HERA01.H1.FPACK.LOAD.V08501   version 0.85/01

CMZ library       : HERA01.H1.FPACK.CMZ           version 0.86/00

Card image file   : HERA01.H1.FPACK.CAR08600

Correction cradle : --- (This version is without history)

CMZ kumac file    : HERA01.H1.FPACK.KUMAC

Examples          : HERA01.H1.FPACK.S
```

```
Machine version specifier          Explanation
-------------------------------+------------------------------------
     IBMMVS                     IBM main frame MVS system (WFIBM)
     IBMDESY                    IBM MVS system at DESY (WFIBM)
     IBM                        IBM main frame VM system (WFIBM)
     VAXVMS                     VAX (WFVAX)
     DECS                       DEC station (WFDEC)
     APOLLO                     APOLLO (WFIEEE)
     MACII                      Macintosh II (WFIEEE)
     ALLIANT                    ALLIANT (WFIEEE)
     SUN                        SUN (WFIEEE)
     SGI                        Silicon Graphics, Inc. (WFIEEE)
     HPUX                       HP Risc Station (WFIEEE)
     IBMRT                      RS 6000 (WFIEEE)
     MIPS                       MIPS (WFIEEE)
     ALPHA                      ALPHA (64 bits addresses) (WFIEEE)

     UNIX                       Common flag for all UNIX machines
```

```
Additional specifier               Explanation
-------------------------------+------------------------------------
     DESY                       DESY specific parts
     BIRM                       Birmingham specific parts
     NET                        Network support
     SHIFT                      SHIFT interface
     FATMEN                     FATMEN interface
```

FPSERVER	FPACK SERVER parts

Patches	Explanation
FPACK_MACROS	Sequences
TEXTUTIL	Utility routines
CONVERT	Conversion routines
FALLIO	Input/output routines
FINPUT	Input routines
FOUTPUT	Output routines
FTEXTIO	Text input/output routines
DEVELOP	Initialization routines
FBANKS	Index routines
TPRETER	Language interpreter routines
FINOUT	Input/output routines
KEYEDACC	Keyed access routines
DIRCTORY	Direct access routines
FPARAM	Language interpreter routines
FSELECT	Record select routines
FINQUIRE	Inquire and statistics routines
FTIME	Time routines in FORTRAN
CTIME	Time routines in C
FBPACK	FB-package routines
FATMEN	FATMEN interface routines
CINTERF	C-FORTRAN interface routines
CINOUT	Input/output routines for networking
CNET	Networking routines
FPSERVER	FPACK server routines
FPACK_ASM	Assembles routines (DESY specific)
FPACK_IBMVM	Special routines for IBMVM

Chapter A

A.1 General introduction

FPACK is a set of FORTRAN77 and C programs for general input/output of data blocks. Important features of FPACK are

* automatic wordformat conversion, and
* record selection options.

FPACK is a stand-alone package and may be used for all kinds of data. It is used for all IO by the BOS system and the LOOK system.

Machine-independence of word formats:
FPACK files are machine-independent. This means that tape files written on some computer A can be read on a computer B, which may have a different internal representation of data, without any special action. The same is true for files transmitted over networks between different computers. FPACK files carry for each physical record a description of the type of the data words, and this information is used for a conversion. The FPACK routines recognize automatically, if a conversion to the internal data representation has to be made during input. On output the user may specify one of four possible data representations on the file; these are:

```
IEEE     for workstations (UNIX)
IBM      for IBM main frames
VAX      for the VAX family
DEC      for DEC stations
```

Machine-independence of files has a consequence for the range of floating-point numbers, since different floating point representations have different ranges of numbers. It is recommended to restrict the range of floating point numbers stored with the FPACK to the range

```
+ or - 10**(-36) up to 10**36.
```

Binary and text mode:
Usually FPACK files are in BINARY mode, which is the more economic mode with respect to speed and to space (in FORTRAN this is called UNFORMATTED). FPACK files may generally be also in TEXT mode (in FORTRAN this mode is called FORMATTED) and these contain the data converted to a character representation, requiring about twice the space on output media; in addition reading and writing is much slower. For small data sets this mode may have some advantages: the files can

can be edited with a text editor and are thus directly readable, and they can be sent as text files over networks.

Logical (user) records:
The logical records (records, as seen from the user) may have any length. Physical records of binary files have a fixed length, and this length should be large for economical reasons. Some computers add control words to certain types of physical records; this is avoided within FPACK and future computer versions should avoid this too, since it can make files computer-dependent. TEXT files have usually a small fixed record length, for example 80 bytes corresponding to the screen width of a terminal. For economical reasons blocking of these small records to at least a few 1000 bytes is recommended on certain computers.

Physical records:
Physical records are of fixed length, without control words (this record format is called record format RECFM=F meaning "fixed unblocked" on IBM main frames) for binary sequential or direct access. The number of bytes of a physical record should be a multiple of 360 for reasons of compatibility with machines, which have a word length different from 32 bits. On some machines (UNIX systems) actual reading and writing is done with routines in the C language. Blocking of records is done inside FPACK: a physical record may contain several logical records, and a logical record may be split over several physical records. As a consequence of this blocking usually logical records do not start in new physical record. By a special option (record separation) however the start of a new physical record for each logical record can be forced.
The authors recommend to use the same physical-record length for data on disks and cartridges/tapes; this allows a simple efficient copy and especially a staging of files from cartridge to disk.

Networking:
A server (FPSERVER) is part of the FPACK. This server allows to access in a simple and efficient way files on different computers. The server is written in the C language and makes use of standards like RPC (Remote Procedure Calls).

File reference:
In FPACK the reference to a file is by a symbolic file name, which can be defined by the user and is a character string of up to 16 characters, for example BOSINPUT. This is in contrast to FORTRAN, where the reference to a file is by the unit number, for example in a statement like READ(UNIT=1). The symbolic file name has to be distinguished from the file_name, under which the file is actually created and catalogued on the computer. The symbolic file_name is called dataname.

Record selection and record header:
FPACK provides options to select certain records; in order to allow this selection each logical (user) records has a short header (data record header) known as the RECORD KEY. The record key allows to access certain records required by the user in an efficient way, and this is the reason for the introduction of the record key. It also allows to use standard utilities for SORTING records.

Two different cases are distinguished, and these are explained in an example. Assume that a file contains records of different type, representing either calibration data for a detector component, or physics events. The user may wish to find calibration data, valid for a certain run or he/she may wish to find a certain physics event (run and event number known). The user may also be interested in all events on the file with a certain characteristic, which are candidates for a specific subprocess.

These two cases represent different requirements and both are supported by the FPACK and are based on the record key, which has two parts.

One part of the RECORD KEY, as seen by the user, is

Record name	string of 8 characters	RECNAME
Record number A	integer	NUMRA
Record number B	integer	NUMRB

and this is meant as a unique identification of the record in the file. Usually there should be no two records on a file with the same identification in these three items.

The other part of the RECORD KEY is called classification. A record can be assigned by the user to one class ICLASS or to several classes, which are numbered 1 ... 30. Internally the assigned classes are stored as bit information in one integer word:

Classification	integer	ICL

Usually there will be many records on a file assigned to a given class.

The actual RECORD KEY contains in addition to the items explained above another item (total length of 20 bytes), which is a record counter, necessary to distinguish parts of a user record in different physical records.

Example for the RECORD KEY:
An example for the first part of the record key is the standard assigment used for data on events in BOS applications of FPACK:

```
Record name       =  'RUNEVENT'
Record number A   =   run number
Record number B   =   event number
```

Although the assigment of classes is arbitrary, it is recommended for physics experiments to divide the classes into three parts, according to the scheme:

ICLASS	classification
1 ... 10	online (trigger, filter)
11 ... 20	offline
21 ... 30	private analysis

Keyed access files:
A special IO-mode, called keyed-access, is supported on some computers, providing certain options, which otherwise would require a large amount of coding on the user side. Within FPACK, the record key explained above is utilized, and among the options available by this special IO-mode are the following:

* deleting records,
* replacing records,
* reading of data records in ascending or descending order of the key.

These options support data-base-like applications.

Ordered access files:
FPACK allows to create the files with special internal structure. They have some kind of internal event directory or index inside them, which contains only information on the record keys and information on the position of the logical record on the file. This index allows a fast access to each record from certain places. The reading of a logical record, knowing its position on the file is realized by opening the file as FORTRAN direct access file. Ordered access files support all the options, supported by keyed access files:

* deleting records,
* replacing records,
* reading of data records in ascending or descending order of the key.

Regardless of the access method the record structure and the basic calling sequences for IO supplied with FPACK are identical. This allows for example to copy a file from keyed-access to a tape, and back to keyed-access, without a loss in functionality.

Index files:
FPACK allows the creation of files containing only record keys and references to other files, where real data records reside. The index file can be created by special procedure (see B.2.4) and after creation can only be used for reading the real data (not for writing it). Switching between an index file (to get the record key) and data files (to get the data) is done automaticaly and hidden for the users.

The structure of FPACK records

In the remaining part of this chapter the structure of FPACK calls and records is explained. The reader may skip this part, since the knowledge of the internal structure is not necessary for a successful use of FPACK files and subprograms. A detailed explanation of all FPACK subroutines is given in the FPACK manual.

The data within data records are organized in blocks, which are identified by:

Data block name	string of 8 characters	NAMDB
Data block number	(any) integer	NUMDB

(For BOS data the data block name and number are the BOS bank name and number). There may be any number of data blocks within a record, and a data block may have any length. A data block is preceeded by a header, which contains, in addition to data block name and number, a description of the word format within the block, and two integers, NCOL and NROW. They can be used to describe the column and row number of a data table. Within the FPACK these two numbers are treated as comment; the actual length of the data block may differ from the product NCOL*NROW.

Reading data records:
In order to read a data record three subroutines are provided by FPACK. The first one (FRKEY) allows to read the key of the next record, the data are not yet read by this call. On the basis of the key the user may decide either to read the key of the next record, or to read the data blocks of the current record. In order to read the data blocks there are two subroutine calls (FRHDR and FRDAT): one to read the data block header, and one to read the data. For standard applications like BOS and LOOK there are subroutines (FRBOS and FRLOOK), which allow to perform by one user call to read and to store all the data. The special structure of FPACK allows the user to read the key of records by FRKEY and to read the data afterwards with the FRBOS and FRLOOK calls.

Writing data records:
For writing data records there are three subroutines (FWKEY, FWHDR, and FWDAT), equivalent to the reading routines explained above. They define the record key, and add the data block header and the data to the current record.

FPACK logical records represent a stream of bytes, with a structure as shown below:

A record header ...

```
RECORD KEY          =   Record name             RECNAME
                        Record number A         NUMRA
                        Record number B         NUMRB
                       [Record counter          RECCOUNTR]
                        Classification          ICL
```

followed by pairs of the following items ...

```
DATA BLOCK HEADER   =   Name of data block      NAMEDB
                        Number of data block    NUMDB
                        number of columns       NCOL
                        number of rows          NROW
                        block format            FORMAT


DATA BLOCK          =   Data (without control information)
```

The stream of bytes is mapped to the fixed record-length structure of FPACK files, adding some control information within the physical records. Especially the record header is repeated each time a new physical record has to be used. More details are given in the Chapter B of this manual.
The record counter given in parenthesis [] above is normally not visible to the user, and is defined only for keyed access and files with RECSEP option (see under OPEN). The values of the counter are 0, 1, ... for the physical records, belonging to one logical record, and for example allow to sort records with a standard SORTING utility.

A.2 The language interpreter

A.2.1 Function of the interpreter

Powerful and flexible program packages need adequate steering options. One possibility of steering is by statements of an easy-to-learn programming language, where the statements are analysed by an interpreter and are immediately executed. This very flexible approach has been chosen for FPACK. The present scope of the interpreter is rather limited, it is just used to steer FPACK functions. Extensions are possible towards an analysis language, but no detailed planning is done at present. The interpreter syntax is similar to the FORTRAN syntax, with modifications to make the syntax well adapted to interactive work at a terminal or workstation. The source form is identical to the FORTRAN 90 rules.

An important property of the executable language interpreter statements is the coexistence with FORTRAN programs. For example one can use calls to FORTRAN subroutines to read records, and execute the interpreter statement REWIND DATANAME for the same file.

The statements used for steering FPACK functions start by certain keywords, which specify the basic operation, and include a variable number of further parameters. The keywords are:

```
OPEN           to open a file and declare the characteristic
RSELECT        to declare data selection criteria for a file
SELECT         to declare data selection criteria for a file
REWIND         to rewind a file
CLOSE          to close a file
FILECOPY       to copy a file to another file
IDXCOPY        to copy entries from a file to another file
IOSTATISTICS   to print current IO statistics
PRINT          to define the print level
TIMESTOP       to set a time parameter
END            to return from FPARMR (see below)
```

The statements can either be given as argument in a call (FPARM) or can be read from a text file (FPARMR). In both cases the statements are processed in the same way: the statement is analysed and executed.

```
             ---------
   CALL FPARM(STATEMENT)  process statement


              ---
   CALL FPARMR(LUN)       read statements from file and process
                          until END-statement


                  ----
   CALL FERMES(MSG,IFLG)  get last message, reset internal message text
               ---        If input parameter IFLG
                               = 0 ... MSG = normal message
                               = 1 ... MSG = error message
```

In case of an error an internal message line is defined. The user may get this message lines to an user array by a call (FERMES); the internal message line is reset at this call. If more than one error occurs, only the last one is stored in the message line. In batch jobs by default the print level is set to 1, which means that statements and diagnostics is printed. Batch programs (FSEQR) stop automatically in case of open errors.

Remark

For the subroutine FPARMR, the most frequently used unit number is 5, the standard input unit. If the user wishes to use a different unit e.g. CALL FPARMR(44), the text input file can be opened for example with

```
  CALL FPARM('OPEN TXTINPUT UNIT=44 FILE="FPARMR.DATA" READ TEXT')
```

or by the user himself.

A.2.2 Source form of interpreter statements

Source form rules

The source form for statements is free, the rules are identical to the rules in FORTRAN 90. Statements are written in one or in a sequence of lines. The special characters "!" and ";" outside character context, and the special character "&" as last (or eventually first) outside a comment have a special significance for the statement separation or continuation.

Within the source text BLANKS are SIGNIFICANT, in contrast to FORTRAN. For example the integer constant thousand has to be given as 1000 (while 1 000 would mean the number 1 followed by the number 0) and the keyword open has to be written without blanks as OPEN (while Fortran would allow to write O P EN). Special characters (like /) act as separators.
In text-constants, where the text is either
          within apostrophs or quotation marks
(see also under constants), upper- and lower-case characters are distinguished, and all special characters are allowed. For text constants given without apostrophes/quotation marks including keywords upper- and lower-case characters are equivalent. Thus OPEN and open and Open is equivalent. Keywords do not countain special characters, only letters and numbers.

Commentary:
The character "!" initiates a comment except when it appears within a character context; the comment extends to the end of the source line. Lines with a "*" in character position 1 are an additional form of commentary. Lines containing only comment or blanks are ignored and can appear anywhere.

Statement separation, multiple statements:
Multiple statements in one line are allowed. The character ";"

separates statements on a single source line except when it appears within a character context. Statements containing no characters or only blanks are ignored.

Statement continuation:
Outside of a comment, the character "&" as the last nonblank character on a line signifies that the statement is continued on the next line (a comment cannot be continued). If the first nonblank character on the next line is also "&", the statement continues at the next character position following the "&"; otherwise, it continues at character position 1. If a character context (other than a comment) is being continued, the "&" signifying continuation cannot be followed by commentary and the continued portion must begin with a "&" . If the continuation is not within a character context, the "&" signifying continuation may be followed by commentary.

Examples for statements:

```
   OPEN BOSINPUT    FILE="F99ABC.DATA"                         ! input file
   OPEN bosoutput   FILE="F99ABC.MYDATA"  WRITE  &
                        RECL=23400 STATUS=NEW
   OPEN BOSinput    file = "F99ABC.TEST.CHARM.MO&
      &CA.RUN123"                         ! note continuation character
*  this is a legal comment statement
   REWIND BOSOUTPUT
   CLOSE BOSINPUT;  CLOSE BOSOUTPUT        ! two statements on one line
```

Constants

Numerical constant are specified in the FORTRAN style and need no detailed explanation, except that, as mentioned above, a blank is a separator. Integer and real constants are given in the examples below.

Character constants:
these are a sequence of characters, delimited either by apostrophes

'text'

or alternatively by quotations marks

"text".

As an alternative a text constant can also be given without quotation marks or apostrophes, like

text

if the meaning is clear from the context. In the latter form all letters are converted to upper-case, and the text may not contain any special character (it would act as a delimiter).

An apostroph character within a character constant delimited by apostrophes is represented by two consecutive apostrophes (without intervening blanks); in this case, the two inner apostrophes are counted as one character. Similarly, a quotation mark character within a character constant delimited by quotation marks is represented as two consecutive quotation marks and the two quotations marks are counted as one character. In character constants upper and lower case characters are allowed, and are not modified (note that for text, which is not within apostrophes or quotes, upper and lower case characters are allowed, but are interpreted as if they are upper case characters).

Examples for constants:

```
         473     ! integer (no dot, optionally with sign)
        - 11
         +56
         -11

      -12.78     ! floating point (with dot, and/or with
      +1 E-6     !   exponent, optionally signed)
      +1,E-6     !   two numbers plus text: 1 and E and -6

      'PEAK'     ! character
        Peak     !       (converted to PEAK)
    'DON''t'     !       (means DON't)
    "Don'T"      !       (means Don'T)
```

Groups of constants:

In some statements groups of constants are accepted, either integer, real constants or character constants (with either apostrophs or quotation marks). The constants may be separated by commas or by blanks. The number of constants for each type is limited to 500.

In the case of integer constants ranges can be specified using the special character colon(:), in the forms

```
      constant1 : constant2           (lower and upper limit given)
      constant1 :                     (upper limit infinity)
                : constant2           (lower limit 1)
```

In these cases a comma may be necessary as a separator instead of blanks (see examples 2 and 3 below).

Examples for group of constants:

```
         1, 4:10 13:15, 21:           ! integer
```

```
11:, 5                        ! means 5, and 11 to infinity
11:  5                        ! wrong, it means range 11:5
3.14, 6.28 9.24               ! real
'RUNEVENT' 'CALIBRA'          ! character
```

A.3 Statements related to input/output

A.3.1 The OPEN statement

```
The OPEN statement for a file has two functions: it (1) defines the
characteristics of the file and (2) executes the actual OPEN for the
data set. Once a file has been opened, and selections can be defined,
records can be read or written, and the file can be rewound. After
these operations it should be closed by a CLOSE statement.  In special
applications the user may want to execute his own OPEN procedure, and
thus to only the function (1) of above; this special case is indicated
by the option NOOPEN. The OPEN statement allows to define several data
files for sequential reading. They will be read one after another.  It
allows also to define intervals of files for reading ( see FILE option
description ).

General form  of the OPEN statement for READ access
   to an existing binary file:

   OPEN
      symbolic_file_name                      text
   or UNIT = unit                             integer constant
   or symbolic_file_name  UNIT = unit         text and integer constant

      FILE = list_of_file_names               text (within apostrophes
                                              if case-sensitive or if
                                              special characters)
      HOST = host_name                        text

In this example the symbolic_file-name is specified as BOSINPUT and all
further reference to the file has to be specified using this name.  The
(FORTRAN-) unit number is not given in this example, a free number is
selected and assigned internally to the symbolic_file_name BOSINPUT.
The keyword HOST can be used to specify the host machine, where the
file is residing; if omitted or if hostname = LOCAL, the file is
assumed to reside on the computer executing the program.

The full syntax for the OPEN statement is as follows:

   OPEN
      symbolic_file_name              text
   or UNIT = unit                     integer constant
```

```
or symb.file_name UNIT = unit     text and integer constant

   FILE = file_name               text, within apostrophes or
                                  double quotation marks if case
                                  sensitive or if it contains
                                  special characters

   HOST = host_name               text

   RECL = record_length           in bytes, integer constant, for
                                  binary mode a multiple of 360

   BLFACTOR = factor              integer constant, blocksize factor
                                          (for TEXT and output only)

   BUFNO = number_of_buffers      integer constant

   NREC = recnum                  integer constant, primary space,
                                  number of physical records

   NREC2 = recnum2                integer constant, secondary space,
                                  number of physical records

   SPLITMB = number of MB         integer constant, limit for the
                                  length of the output file
                                            (for output only)

   SPLITEV = number of events     integer constant, limit for the
                                  number of events, written to the
                                  output file  (for output only)

   [ACTION =]  READ or WRITE or READWRITE or MODIFY

   [ACCESS =]  SEQ or DIR or KEYED or SPECIAL or ORDERED
                               (DIR and SPECIAL not yet implemented)

   [STATUS =]  OLD or NEW or SCR


   [FORM =]    TEXT or BINARY

   [WORDFMT=]  WFLOCAL or WFIEEE or WFIBM or WFVAX or WFDEC
                                            (for output only)

   RECSEP      logical records will be separated
                                            (for output only)

   NOCOMP      switch off the compacting of data written on the
               cartridge for export         (for output only)
```

```
SKIPCOREV   skip corrupted events        (for input only)

DUNIT = data_unit_number                 integer
            for reading cartridges using index files

DEFER       IBM catridge access (DESY mainframe only)

DDEFER      flag (DESY mainframe only), which forces FPACK to
            read cartridge files, which are referenced in the
            event directory file, directly from the cartridges.
            (no staging).

VOLSER = list_of_volume_names            text
            IBM private catridge access (DESY mainframe only)
            (e.g. cartridges for export)

STAGE       copy a file from a cartridge to a disc
                                         (for input only)

DSTAGE      flag (DESY mainframe only), which forces FPACK to
            stage cartridge files, which are referenced in the
            event directory file (no direct read from the
            cartridges).

UNIXTAPE    I/O to tapes (for example EXABYTE) on UNIX machines
            (not implemented for APOLLO and HPUX yet)

NOOPEN      file connected, NO OPEN within FPACK, open have to
            be done by the user (for very special cases)
```

Examples for open statements:

```
   OPEN BOSINPUT FILE=F99ABC.H1SIM.I0010008       ! open existing file
Assumed defaults are: READ SEQ OLD BINARY

   OPEN BOSINPUT FILE="F99ABC.DATA1,F99ABC.DATA2,F99ABC.DATA3"
                          ! concatenation of three files for reading

   OPEN BOSINPUT FILE="F99ABC.DATA.A17-C32"
Allows to read a set of files with the names
DATA.A17, .A18, ..., .A99, .B00, ..., .B99, .C00, ..., DATA.C32

   OPEN BOSOUTPUT FILE=F99ABC.H1SIM.I0010009 WRITE  &
        RECL = 23400                              ! open new file
Assumed defaults are: NEW BINARY WFLOCAL

   OPEN CARTFILE UNIT=1 FILE="F99ABC.DATA.ON.THE.CART" STAGE
The file will be staged and allocated.
```

```
Detailed explanation of the OPEN statement:
-------------------------------------------



   OPEN
       symbolic_file_name                      text
    or UNIT = unit                             integer constant
    or symb.file_name  UNIT = unit             text and integer constant
```

```
If the dataname is specified, it has to be the first item  after  OPEN.
The (FORTRAN) unit number may be specified after the dataname; if  not,
a free unit number will be assigned  automatically  to  the  file.  The
standard system input and output units (usually 5 and 6) are  known  to
FPACK. The unit numbers assigned to the user file will start at  number
11. Unit numbers not to be assigned  have  to  be  specicified  to  the
system by : CALL FRESER(LUN). In some cases  it  may  be  necessary  to
specify the unit number (for example if a tape unit has  been  reserved
in a batch job for a certain unit).




The following options for the OPEN statement each  require  a  keyword,
followed by an equal sign(=), followed by a constant (text or integer).

      FILE = list_of_file_names                 text

The file_name has to be specified for old or new permanent  files,  but
not for scratch files. On certain machines  the  file_name  has  to  be
enclosed  in  quotes,  otherwise  all  characters  are   converted   to
upper_case characters.


Multifile input :
-----------------

Multifile input is allowed only  for  sequential  files  and  only  for
reading. The FILE parameter in the OPEN command can contain a  sequence
of file names, separated by commas. It means, that several  files  will
be read one after another.
For example,
   OPEN BOSINPUT FILE="F99ABC.DATA1,F99ABC.DATA2,F99ABC.DATA3"
opens the file DATA1 for reading and  manages  automatic  switching  to
file DATA2 after an end of  file  DATA1  has  been  reached,  and  then
switching to DATA3 file.

If the file name  ends  with  a  sequence  Point-UppercaseLetter-Digit-
Digit-Minus-UppercaseLetter-Digit-Digit, it is  treated  in  a  special
way, as file interval description. For example,
```

```
   OPEN BOSINPUT FILE="F99ABC.DATA.A17-C32"
```

doesn't open the file F99ABC.DATA.A17-C32, but opens the file DATA.A17 and manages automatic switching to file DATA.A18 after end of file DATA.A17 has been reached, then switching to DATA.A19, then DATA.A20, ..., DATA.A99, then DATA.B00, ..., DATA.B99, DATA.C00, ..., DATA.C32.

Intervals may be freely used in a sequence. For example,

```
   OPEN BOSINPUT FILE="A.BABY,A.P17-R32,AN.ADULT,X.A00-A12"
```

Files in a sequence, and even files in one interval may have different record length.

```
      HOST = host_name              text
```

The host_name has to be specified, if the file is not on the machine executing the program.

Further parameters for new files:

```
      RECL = record_length          in bytes, integer constant
```

The record length can be specified for a new file; a default value is assumed, if the record length is not specified. For F files it is the length of the binary record and should be a multiple of 360 (compatibility between computers with different wordlength); the default value is installation-dependent. For an existing input file the length is taken from the file itself. For text files RECL is the record length of one line of text, with default 80. For certain machines IO is speeded up by using a large blocking. The user may specify a blocking factor by

```
      BLFACTOR = factor             integer constant
```

to specify the total record length as "factor * record_length". The factor is ignored on machines, where blocking is not necessary or possible. Example for the IBM/MVS: RECL=80 BLFACTOR=10 corresponds to RECFM=FB,BLKSIZE=800,LRECL=80.

```
      BUFNO = number_of_buffers     integer constant
```

Parameter BUFNO gives the number of buffers for I/O operations. The default value is 2.

For new files the user may specify the size of the new file in terms of the number of records (NREC). On certain machines it is possible to specify a primary and a secondary quantity, which can be specified as NREC and NREC2 in

```
      NREC = recnum                 integer constant, primary space,
                                    number of physical records
```

```
      NREC2 = recnum2              integer constant, secondary space,
                                   number of physical records
```

This information NREC and NREC2 is ignored on certain machines.

Parameter SPLITMB gives the limit for the maximum length of the output file. First SPLITMB megabytes will be written into the file "F99ABC.DATA.A00" (if FILE="F99ABC.DATA" has been given). After reaching this limit, the current logical record will be written to the end, the file will be closed and the next file "F99ABC.DATA.A01" will be opened without a break in the program. Further filenames have the extensions .A02, .A03, ... , .A99, .B00, .B01, ... , .Z99.

```
      SPLITMB = number of MB       integer constant, limit for the
                                   length of the output file in MB
                                             (for output only)
```

If you use SPLITMB parameter, and file name, given in the FILE parameter, ends with the sequence Point-UppercaseLetter-Digit-Digit, then appending of .A00 to file name is not done.

The command

```
   OPEN FPACKOUT WRITE SPLITMB=200 FILE="F99ABC.DATA"
```

will produce files F99ABC.DATA.A00, DATA.A01, ..., DATA.Z99.

The command

```
   OPEN FPACKOUT WRITE SPLITMB=200 FILE="F99ABC.DATA.B17"
```

will produce files F99ABC.DATA.B17, DATA.B18, ..., DATA.Z99.
In a very special case, when you need to produce the files F99ABC.DATA.B17.A00, DATA.B17.A01 ..., you must code FILE="F99ABC.DATA.B17.A00" instead of FILE="F99ABC.DATA.B17".

Parameter SPLITEV gives the limit for the maximum length of the output file. First SPLITEV logical records will be written into the first file according the scheme desribed in SPLITMB.

```
      SPLITEV = number of logical records        integer constant,
                                   limit for the length of the
                                   output file in logical records
                                             (for output only)
```

It is not allowed to use the parameters SPLITMB and SPLITEV together in one OPEN command.

```
      [ACTION =]  READ or WRITE or READWRITE or MODIFY
```

By default READ is assumed. The action parameter is in certain installations important for data security. It is also observed within the F package. This means, that for actions values READ or READWRITE, overwriting of existing data will not be possible (READWRITE would allow to add data); for example writing call after a rewind would not be executed. The option WRITE means writing from the first record (no reading), while MODIFY would allow overwriting or replacing of records.

```
[ACCESS =]  SEQ or DIR or KEYED or SPECIAL or ORDERED
```

For existing files the access mode is determined automatically, with the exception of ORDERED files, where it must be coded explicitly in an OPEN statement. The keyed access is only possible on IBM main_frames at present, which require a special procedure for the creation of new files (not described here). By default SEQ (sequential) is assumed. For text files only SEQ is possible.

```
[STATUS =]  OLD or NEW or SCR
```

By default OLD is assumed for all ACTIONs except for ACTION = WRITE, where NEW is assumed by default.

```
[FORM =]    BINARY or TEXT
```

By default BINARY format is assumed.

```
[WORDFMT=]  WFLOCAL or WFIEEE or WFIBM or WFVAX or WFDEC
```

This option is only necesssary for output of binary FPACK files and it specifies the word format of the data, written to a file. By default the local word format (WFLOCAL) is assumed, which is the fastest of the options for writing.

```
RECSEP
```

If this option is specified for an output file, then logical records will always start with a new physical record. This options is automatically used for keyed_access and ordered_access files. For sequential files it allows to apply SORT procedures to files with sorting according to the record key.

```
NOCOMP
```

This option is valid only for cartridge output files. It switches off the automatic compacting of information written on the cartridge. This option should be used only for export cartridges.

```
SKIPCOREV
```

This option is valid only for input files. It makes it possible to skip corrupted events. These are events which are not written to the end correctly since the program crashed or stopped for some other reason.

DUNIT = data_unit_number   integer

The DUNIT parameter has to be specified if the FILE="index_file" contains references to files on cartridge. The data_unit_number (e.g. 15) has to correspond to the unit number given in the standard DD card: //G.FT15F001 DD UNIT=(CART,,DEFER),DISP=

DEFER

Permits cartridge access (see App. H). All parameters are defined in FPARM call. Only a standard DD card has to be in JCL :
//G.FT15F001 DD UNIT=(CART,,DEFER),DISP=

DDEFER

may be specified if the FILE="index_file" contains references to file on STAGE disk pool (during creation of index file, the data file has been resided on stage-disk), but you want to force FPACK using the cartridge copy. In this case FPACK is forced to read from the cartridge directly (no staging).

VOLSER=list_of_volume_names   text

Permits access to private cartridges on the DESY IBM (e.g. cartridges for export). This option is valid for input and for output files as well. It has to be used together with the parameter DEFER and a standard DD card has to be in JCL :
//G.FT15F001 DD UNIT=(CART,,DEFER),VOL=SER=BBBBBB,DISP=
The volume_name BBBBBB should start with the character B (reserved for H1 cartridges), but should NOT correspond to an existing cartridge label. The list_of_volume_names has the same syntax rules as the list_of_file_names. The users have to take care that the number of file_names and volume_names for input are equal.

STAGE

Permits cartridge access for input only. It is the only one possibility to access cartridges on the DESY IBM remotely. No DD card is necessary. The whole file on the cartridge will be copied to the special disc. If such a copy already exists, it will be allocated. No further copies will be unnecessarily created. Staging can work only in batch. If the user will use staging interactively (for example in LOOK), he has to type NEWLIB command STAGEI LDS=F99ABC.DATA.ON.THE.CART before starting H1ED. To open this file, the user has to type

```
FPARM 'OPEN UNIT=1 FILE="F99ABC.DATA.ON.THE.CART" STAGE'
```

DSTAGE

may be specified if the FILE="index_file" contains references to data file on cartridge, but you want to force FPACK to make staging before opening (no direct read from the cartridges).

UNIXTAPE

I/O to tapes (for example EXABYTE) under UNIX. If this keyword is used, then RECL must also be specified. The user has to position the tape properly before opening the file. The FILE parameter must be set to the name of the tape driver (e.g. "/dev/nrtape" on DESY SGI).

NOOPEN

The actual OPEN is not executed, if the keyword NOOPEN is present.

A.3.2 The RSELECT statement

The RSELECT (Record SELECTion) statement can be used to define certain selection criteria for the data of a FPACK file and certain other options. It is possible to use the RSELECT statement several times for the same file, for example to modify certain selections or to add further selections. The select options are valid for input and for output (except "RECORD/NORECORD", which is only valid for input), and are accepted only for a file, which is already opened Usually there are two possibilities for each of the options, one starting with a "NOT", meaning negation.

The symbolic file name or dataname has to be specified as first item after the keyword RSELECT.

```
RSELECT  data_name  ... (selections)
```

More than one keyword can be given on one RSELECT statement, alternatively the selection can be done in more than one RSELECT statements for the same file. If the same keyword is used again, than the previous selection for that keyword is overwritten. A keyword and its negation can not be used simultaneously, for example if BLOCK has been used already, the keyword NOTBLOCK overwrites the previous BLOCK selection.

All previous selections are reset by the specification of the keyword

RESET

in a RSELECT statement (where the position of the RESET keyword is irrelevant).

Several selections make use of the record key, which contains a character string as record name, and two integers; for example

```
            Record name         =  'RUNEVENT'
            Record number A     =   run number
            Record number B     =   event number.
```

In the H1 experiment there are the two record names 'RUNEVENT' (for event data) and 'RUNDATA' (for rundependent non-event data).

(1) A selection, with keyword RECORD or NOTRECORD, is possible on the record number (first record on the input file has number 1, second has number 2 etc.):.

```
      RECORD      = record numbers (group of integers), or
      NOTRECORD   = record numbers (group of integers).
```

For option RECORD only those record number will be read, which are given within the group of integers, for example

```
   RECORD = 1, 2,  4:9,20:
```

means reading of records 1,2, 4 to 9, and from 20 on. The keyword NOTRECORD means negation.

If a finite set of record numbers is selected, a stop signal is created after reading the record with the highest accepted record number; batch programs (FSEQR) recognize the stop signal automatically. Note that in the example above the set of selected record numbers is not finite (20: means records from record number 20 on are selected).

For output the RECORD option is ignored (the meaning of record number would be unclear in that case). All further options given below apply to read and write operations as well.

(2) A selection is possible, for records with a certain record name, on the basis of the record classification.

```
      CLASS       = record_name, followed by integers
      NOTCLASS    = record_name, followed by integers
                      (integers in the range 1...30)
```

Among records with record names equal to the given recname, records are selected by the selected classes; records with a different record name are not affected by this selection. NOTCLASS is the negation. Several of these selections are allowed within one RSELECT statement. If you use several CLASS/NOTCLASS selections for one record name, then only those records with the given name, which satisfy ALL of the given CLASS/NOTCLASS selections for that name, will be selected. Selections with different record names are independent from each other.

(3) Using the keyword RECNAME or NOTRECNAM, a selection on the basis of the record name(example 'RUNEVENT') is possible:

```
      RECNAME    = record names (text items), or
      NOTRECNAME = record names (text items).
```

For option RECNAME only records with a record name in the given group of record names are read or written. The selection is reversed for the option NOTRECNAME.

(4) Using the keyword RECRANGE or NOTRECRANGE, a selection is possible on the basis of one record name and on ranges of the first record number (example 'RUNEVENT' and run number):

```
      RECRANGE    = record name, followed by numbers
      NOTRECRANGE = record name, followed by numbers.
```

For keyword RECRANGE and records with the given recname only those records, where the first numerical key is within the group of integers, are read or written. All records with different record names are not influenced by the parameter. NOTRECRANGE means negation. Several of these selections are allowed within one RSELECT statement. You can use several RECRANGE/NOTRECRANGE selections with one record name, but it's not recommended, for in that case only those records with the given name, which satisfy ALL of the given RECRANGE/NOTRECRANGE selections with that name, will be selected.

(5) In some applications, the second integer of the record key (see above, Record number B) is a time_stamp (UNIX time = seconds since 1. 1. 1970 GMT). A selection on ALL record names on a file is possible based on the date

```
      BEFOREDATE  = date (in the integer form YYMMDD), or
      AFTERDATE   = date (in the integer form YYMMDD).
```

Records before the given date (BEFOREDATE) or after the given date (AFTERDATE) are selected; noon (12:00 hours) is assumed in both cases. Note that this selection works on all records on the file, and is meaningful only, if the second number (number B) in the record key really is the UNIX date of the record (for example the FB package for data_base like applications). You can use BEFOREDATE and AFTERDATE selections simultaneously, selecting the records in some date interval.

(6) A selection on data blocks within records is possible, for records of a certain name, on the basis of the data block name:

```
      BLOCK      = record_name, block names, or
      NOTBLOCK   = record_name, block names.
```

For option BLOCK only those blocks with a block name (for BOS: bank name) in the given group are read or written, for records with the given record name. The selection is reversed for the option NOTBLOCK. Several of these selections are allowed within one RSELECT statement. You can use several BLOCK/NOTBLOCK selections with one record name, but it's not recommended, for in that case only those blocks in a record with the given name, which satisfy ALL of the given BLOCK/NOTBLOCK selections with that name, will be selected.

(7) A selection on specific records is possible on the basis of the complete record key:

```
      RECKEY     = record_name, recnumA, followed by group of integers
      NOTRECKEY  = record_name, recnumA, followed by group of integers
```

If for example the record name is 'RUNEVENT', and the two numbers of the key are the run # and the event #, then

```
      RECKEY     = 'RUNEVENT' 3033 12,24:28
```

means the selection of events 12 and 24 to 28 from run 3033. Several of these selections are allowed within one RSELECT statement.
The selection is reversed for the NOTRECKEY keyword.
If, for the keyword RECKEY, a record is read with a numbers higher than all selected numbers, a stop signal is created.

(8) A STOP conditions may be defined with the RSELECT statement on the basis of the amount of data read or written.

```
      STOPMB     = megabytes (integer or floating point)
```

A stop signal is created, if more than the specified amount of data is read or written; batch programs (FSEQR) recognize the stop signal automatically.

Summary of the RSELECT statement:

```
  RSELECT data_name   ... selection

      RESET

      RECORD       =  record numbers
      NOTRECORD    =  record numbers

      CLASS        =  record name, classes (integers 1...30)
      NOTCLASS     =  record name, classes (integers 1...30)

      RECNAME      =  record names
      NOTRECNAME   =  record names

      RECRANGE     =  record name, numbers
      NOTRECRANGE  =  record name, numbers

      BEFOREDATE   =  date (YYMMDD)
      AFTERDATE    =  date (YYMMDD)
```

```
BLOCK       =  record name, block names
NOTBLOCK    =  record name, block names

RECKEY      =  record name, number_A, numbers
NOTRECKEY   =  record name, number_A, numbers

STOPMB      =  mbytes (integer or floating point number)
```

Example for RSELECT statement:

```
RSELECT BOSINPUT                                     &
          RECORD   = 1:3, 10:                        &
          CLASS    = 'RUNEVENT'  1 3, 4:7 17         &
          RECNAME  = 'RUNEVENT' 'CALIBRA'            &
          RECRANGE = "RUNEVENT" 4711:4713, 4718

RSELECT BOSOUTPUT                                    &
          NOTBLOCK = "RUNEVENT" 'MODC', 'MODR'
```

Explanation: the file BOSINPUT is an input file with BOS data, the file BOSOUTPUT is the corresponding output file.

According to the RECORD option, the first three records and all the records from the 10th on are read (RECORD).

Only records of classes 1, 3, 4, 5, 6, 7 and 17 are read (CLASS).

Only records with the names 'RUNEVENT' or 'CALIBRA' are read (RECNAME), others are automatically skipped.

Among the records with the title 'RUNEVENT', only those with (run) numbers 4711, 4712, 4713 and 4718 are read (RECRANGE).

The RSELECT statement for the output file contains the NOTBLOCK option, which is used in the example to suppress the output of BOS banks with names MODC and MODR (this example is somewhat artificial; in reality nobody would suppress the output of these banks).

A.3.3 The SELECT statement

The SELECT statement can be used to define certain selection criteria for the data of a FPACK file and certain other options. The symbolic file name or dataname has to be specified as first item after the keyword SELECT.

```
   SELECT  data_name  ... (selections)
```

Characteristics :

----

1) Is easier than RSELECT statement.
2) Much extends the current possibilities of RSELECT.
3) Allows selection of single parameters, for example selection of run_number only (without other conditions).
4) Allows logical operations .AND. and .OR. between SELECT keywords.
   a) Parameters within one keyword will be handled with .OR.
   b) Keywords within one SELECT statement will be handled with .AND.
   c) Separate SELECT statements will be handled with .OR.
   d) Keywords BLOCK/NOTBLOCK will be handled always with .AND. with all other condisions.

The select options are valid for input and for output (except "RECORD/NORECORD", which is only valid for input), and are accepted only for a file, which is already opened.

Usually there are two possibilities for each of the options, one starting with a "NOT", meaning negation.

Syntax of the SELECT statement :

```
      RESET            - reset all previous selections

      RECORD           - record numbers
      NOTRECORD        - record numbers

      RECKEY           - record_name  run_number  event_numbers
      NOTRECKEY        - record_name  run_number  event_numbers

      RECNAME          - record_names (text items)
      NOTRECNAME       - record_names (text items)

      NUMRA            - run_numbers
      NOTNUMRA         - run_numbers

      NUMRB            - event_numbers
      NOTNUMRB         - event_numbers

      CLASS            - classification bit_numbers
      NOTCLASS         - classification bit_numbers

      BLOCK            - bank_names
      NOTBLOCK         - bank_names
```

```
      BEFOREDATE       - date (in the integer form YYMMDD)
      AFTERDATE        - date (in the integer form YYMMDD)

      STOPMB           - megabytes (integer or floating point)
```

(1) RESET
All previous selections are reset by

```
  SELECT data_name RESET
```

(2) RECORD/NORECORD
The selection on the record_numbers is valid only for input. The first record on the input file has number 1, the second has number 2, ...

```
      RECORD      = record numbers
      NOTRECORD   = record numbers
```

For option RECORD only those records will be read, which numbers are given within the group of integers, for example

```
  SELECT data_name RECORD = 1, 2,  4:9,20:
```

means reading of records 1,2, 4 to 9, and from 20 to the end of file.

If a finite set of record numbers is selected, a stop signal is created after reading the record with the highest accepted record number; batch programs (FSEQR) recognize the stop signal automatically. Note that in the example above the set of selected record numbers is not finite (20: means records from record number 20 on are selected).

The keywords RECKEY/NOTRECKEY, RECNAME/NOTRECNAME, NUMRA/NOTNUMRA and NUMRB/NOTNUMRB must not be used twice in one SELECT statement.

If the keyword RECKEY/NOTRECKEY is used, then the keywords RECNAME/NOTRECNAME, NUMRA/NOTNUMRA and NUMRB/NOTNUMRB must not be used in the same SELECT statement.

(These combinations have no physical sense.)

(3) RECKEY/NOTRECKEY
The selection on specific records is possible on the basis of the complete record_key

```
      RECKEY     = record_name  run_number  event_numbers
      NOTRECKEY  = record_name  run_number  event_numbers
```

If for example the record_name is 'RUNEVENT', the run_number is 3033 and the event_numbers are 12, 24, 25, ... , 28.

```
  SELECT data_name RECKEY = RUNEVENT 3033 12,24:28
```

means the selection of events with the name RUNEVENT 12 and 24 to 28 from run 3033. Using of more of these selections within one SELECT statement is not allowed (would have no sense, while they are haldled with .AND.). Several events from two different runs can be selected using two SELECT statements

```
  SELECT data_name RECKEY = RUNEVENT 3033 12,24:28
  SELECT data_name RECKEY = RUNEVENT 3034 72 73 88:102
```

(4) RECNAME/NOTRECNAME

The selection on the basis of the record_name (for example RUNEVENT) is possible

```
      RECNAME    = record_names (text items)
      NOTRECNAME = record_names (text items)
```

For option RECNAME only records with a record name in the given group of record names are read or written. The selection is reversed for the option NOTRECNAME.

(5) NUMRA/NOTNUMRA

The selection on the run_numbers is possible

```
      NUMRA      = run_numbers
      NOTNUMRA   = run_numbers
```

The statement

```
  SELECT data_name NUMRA = 3033 3034 CLASS = 3
```

will select all events from the runs 3033 and 3034 with class 3.

(6) NUMRB/NOTNUMRB

The selection on the event_numbers is possible

```
      NUMRB      = event_numbers
      NOTNUMRB   = event_numbers
```

The statement

```
  SELECT data_name RECORD = :200 NUMRB = 12 24:28
```

will select events 12, 24, 25, ..., 28, if they are within first 200 records.

(7) CLASS/NOTCLASS

The selection on the classification bit_numbers. It is undependent on other keywords and can be logical combined with them.

```
      CLASS       = classification bit_numbers
      NOTCLASS    = classification bit_numbers
```

The statement

```
  SELECT data_name CLASS = 1 3
```
will select events with class 1 or class 3.
The statement
```
  SELECT data_name RECNAME = RUNEVENT NUMRA = 3033 CLASS = 1 CLASS = 3
```
will select RUNEVEVT events from the run 3033 with class 1 and class 3.

(8) BLOCK/NOTBLOCK
The selection on data blocks (bank_names) will be handled always with .AND. with other conditions in the same SELECT statement and all other conditions in all other SELECT statements.

```
      BLOCK      = bank_names
      NOTBLOCK   = bank_names
```

The statement
```
  SELECT data_name NUMRA = 3033 BLOCK = CRJT
```
will select events from the run 3033 and only the banks CRJT will be read.

(9) BEFOREDATE/AFTERDATE
In some applications, the second integer of the record key (NUMRB) is a time_stamp (UNIX time = seconds since 1. 1. 1970 GMT). A selection on ALL record names on a file is possible based on the date

```
      BEFOREDATE  = date (in the integer form YYMMDD)
      AFTERDATE   = date (in the integer form YYMMDD)
```

Records before the given date (BEFOREDATE) or after the given date (AFTERDATE) are selected; noon (12:00 hours) is assumed in both cases. Note that this selection works on all records on the file, and is meaningful only, if the second number (number B) in the record key really is the UNIX date of the record (for example the FB package for data_base like applications). You can use BEFOREDATE and AFTERDATE selections simultaneously, selecting the records in some date interval.

(10) STOP
The STOP conditions may be defined with the RSELECT statement on the basis of the amount of data read or written.

```
      STOPMB     = megabytes (integer or floating point)
```

A stop signal is created, if more than the specified amount of data is read or written; batch programs (FSEQR) recognize the stop signal automatically.

```
Examples for SELECT statement :
-------------------------------


-----------------------------------------------------------------------
1)  The command
SELECT DATAIN RECKEY = RUNEVENT 26222 187 212:214
is the same as
SELECT DATAIN RECNAME = RUNEVENT NUMRA = 26222 NUMRB = 187 212:214

Means : RUNEVENT .AND. 26222 .AND. ( 187 .OR. 212 .OR. 213 .OR. 214 )

-----------------------------------------------------------------------
2)  This command is not allowed
*** SELECT DATAIN RECKEY = RUNEVENT RUNDATA 26222:26228 187 212:214 ***
( only one run_number allowed ), but this command is valid
SELECT DATAIN RECNAME = RUNEVENT RUNDATA &
              NUMRA = 26222:26228 &
              NUMRB = 187 212:214

Means : ( RUNEVENT .OR. RUNDATA ) .AND.
        ( 26222 .OR. 26223 .OR. 26224 ) .AND.
        ( 187 .OR. 212 .OR. 213 .OR. 214 )
-----------------------------------------------------------------------




-----------------------------------------------------------------------
3a)
SELECT DATAIN RECNAME = RUNEVENT
SELECT DATAIN NUMRA = 26222
SELECT DATAIN NUMRB = 187

Means : RUNEVENT .OR. 26222 .OR. 187  ( This has probably no sense. )

3b)
SELECT DATAIN RECNAME = RUNEVENT &
              NUMRA = 26222 &
              NUMRB = 187

Means : RUNEVENT .AND. 26222 .AND. 187

-----------------------------------------------------------------------
4)  The commands
SELECT DATAIN CLASS = 1,3
SELECT DATAIN CLASS = 8
are the same as this one command
SELECT DATAIN CLASS = 1 3 8

Means : 1 .OR. 3 .OR. 8
```

```
-----------------------------------------------------------------------
5)
SELECT DATAIN CLASS = 1 CLASS = 3
SELECT DATAIN CLASS = 8

Means : ( 1 .AND. 3 ) .OR. 8

-----------------------------------------------------------------------
6)
SELECT DATAIN RECNAME = RUNEVENT CLASS = 1,3 &
              BEFOREDATE = time

Means : RUNEVENT .AND. ( 1 .OR. 3 ) .AND. time

-----------------------------------------------------------------------
7)
SELECT DATAIN RECNAME = RUNEVENT CLASS = 1 CLASS = 3,8:10 &
              BEFOREDATE = time

Means : RUNEVENT .AND. ( 1 .AND. ( 3 .OR. 8 .OR. 9 .OR. 10 ) .AND. time

-----------------------------------------------------------------------
8a)
SELECT DATAIN RECNAME = RUNEVENT CLASS = 1 CLASS = 3 &
              BEFOREDATE = time
SELECT DATAIN CLASS = 8

Means : ( RUNEVENT .AND. 1 .AND. 3 .AND. time ) .OR. 8
                                      ( This has probably no sense. )

8b)
SELECT DATAIN RECNAME = RUNEVENT CLASS = 1 CLASS = 3 &
              BEFOREDATE = time1
SELECT DATAIN RECNAME = RUNEVENT CLASS = 8 &
              BEFOREDATE = time2

Means : ( RUNEVENT .AND. ( 1 .AND. 3 ) .AND. time ) .OR.
            ( RUNEVENT .AND. 8 .AND. time1 )    =
        =   ( RUNEVENT .AND. time2 ) .AND. ( 1 .AND. 3 .OR. 8 )

-----------------------------------------------------------------------
9)  The command
SELECT DATAIN RECKEY = RUNEVENT 26222 187 212:214 &
              BLOCK = HEAD CRJT
is the same as
SELECT DATAIN RECKEY = RUNEVENT 26222 187 212:214 &
SELECT DATAIN BLOCK = HEAD CRJT

Means : RUNEVENT .AND. 26222 .AND. ( 187 .OR. 212 .OR. 213 .OR. 214 )
        and from these events only banks HEAD and CRJT
```

```
-----------------------------------------------------------------------
10)  The command
SELECT DATAIN RECORD = 125,:13
will select records 1, 2, ... , 13 and 125
This command is incorrect
*** SELECT DATAIN RECORD = 125 :13 ***
-----------------------------------------------------------------------
```

A.3.4 The REWIND statement

```
   REWIND dataname
```

The file is rewound. The next read afterwards will return the first record of the file.

A.3.5 The CLOSE statement

```
   CLOSE
        data_name                   (text)
     or UNIT = unit                 (integer constan)
     or dataname  UNIT = unit       (text and integer constant)

       [STATUS =]  KEEP or DELETE

        ALL                         closes all opened files opened
                                    within FPACK

        NOCLOSE                     NOT YET IMPLEMENTED
```

The CLOSE statement for a file has three functions: it (1) closes the file (the actual CLOSE is not done, if the keyword NOCLOSE is present), (2) removes all file parameters from the FPACK list, and (3) removes all selections for the file.

The rules to specify the dataname and/or the unit number are identical to the OPEN statement. The status parameter is optional; if not given KEEP is assumed except for a file with STATUS = SCR in the OPEN statement.

A.3.6 The FILECOPY statement

The syntax of the statement is:

```
FILECOPY  from_data_name  TO  to_data_name
```

Records from the input file (from_data_name) are copied to the output file (to_data_name).

Selection criteria defined for the files are observed. They can for example be used to copy a subset of the records.

In this normal copy operation modifications of the record attributes is possible; for example the block length or the word format may be changed. No limitations are imposed on the record sizes (note that if files are copied by reading them into the BOS common and writing from the BOS common, records exceeding the size of the BOS common are lost; furthermore there would be a time overhead because of unnecessary BOS operations).

One-to-One file copy: a faster mode can be selected for complete files, if no selection is done and the record attributes are not changed. This mode, faster by about a factor of three, is selected by the additional keyword ALL in the statement:

```
FILECOPY  from_data_name  TO  to_data_name  ALL
```

If the keyword ALL has been used, concatenation of more files and selection of any information is not possible.

On the DESY IBM you can list a complete JCL member with a standard FPACK COPY job by putting a * into col 1 of the next line:

```
LIST 'HERA01.H1.FPACK.S(#FILCOPY)'
```

In this job the following short program is executed:

```
CALL FPARMR(5)
END
```

The job reads, interpretes and executes statements.

A.3.7 The IDXCOPY statement

The syntax of the statement is:

```
IDXCOPY  from_data_name  TO  to_data_name
```

The input file (from_data_name) can be either data or index file and the output file (to_data_name) is an index file. Entries from the input file are copied to the output index file.

Selection criteria defined for the files are observed. They can for example be used to copy a subset of the entries form the input file into the output index file.

On the DESY IBM you can list a complete JCL member with an example job by putting a * into col 1 of the next line:

```
LIST 'HERA01.H1.FPACK.S(#IDXCOPY)'
```

In this job the following short program is executed:

```
   CALL FPARMR(5)
   END
```

The job reads, interpretes and executes statements.

A.3.8 Further statement

The statements described below set certain conditions. In the routine FSEQR the conditions are checked automatically.

Statements:

```
TIMESTOP   seconds (integer or floating point)
```

The value is stored and used for checks of the time remaining in the job step before a time_overflow. Unfortunately the time_routine which is available for this test has a coarse granularity. It is recommended to specify at least 1.05 seconds for the DESY IBM.

In the standard event processing loop the time between two FSEQR calls will be measured and the average from last five values will be compared with this given value. The greater value will be used for the decision to stop the processing. This should prevent a situation, where is not enough time for writting out the last internal FPACK buffer into the output file (prevent from corrupting the last written event).

```
   PRINT      print_flag (integer)
```

The value of the print_flag is stored. The meaning should be:

```
print_flag  =  0   No printout (quiet)
            =  1   Minimum printout (Errors and warnings)
            >  1   more and more printout
```

```
   IOSTATISTICS
```

Statistics on all opened files is printed.

The status of some conditions may be tested by the subroutine FQCOND.

```
                 -----
     CALL FQCOND(CASET,JARG)     get condition code
                       ----
```

```
     CASET text    JARG value    explanation

      TIME          1 if true     left time less than limit
      STOP         >0 if true     any other stop condition
      OPENERROR    >0 if true     any OPEN error (it will be reset
                                           after the call)
      PRINT         print_level
```

A.4 The FPACK and BOS

All BOS files are in the FPACK format. For standard reading and writing of FPACK/BOS files there is one subroutine BFSEQR. The usage of this subroutine is explained in chapter 4.1. The basic operations within BOS are explained in chapter 4.2. For special application there are additional subroutines explained in Chapter B of this manual.

-------------------------- H1 convention ------------------------------

The convention adopted by the H1 collaboration is explained below

```
(final decision, taken from a computer MAIL by H1KADR from April  3.rd,
1992)

Record keys in the H1 experiment for real data and Monte Carlo):

On event files there can be two types of records

 - All records with event data have the record key
   'RUNEVENT'       run#     event#

 - All records with non_events have the record key
   'RUNDATA'        run#     'fake event number for real data'
                            'UNIX time stamp for Monte Carlo data'

No other records should exist on event files. The event number  or  the
'fake event number/Unix time stamp' make the record key unique!

The RUNDATA and RUNEVENT records can be intermixed on one file i.e. are
not necessary always separated from the data and send to  the  database
during event logging

The non_event records get the BOS flag setting
             REVENT = .FALSE.
             OTHDAT = .TRUE.
These records  can  switch  on  the  ENDRUN/BEGRUN  flags.  No  special
treatement of  these  non-event  records  is  made  w.r.t  reading  and
writing, compared to the event records: the user  specific  choice  for
selecting one or the other should be steered by the FPACK SELECT option
(available in FPACK)

REMARKS
-------
 1.The record key is the master, not the HEAD bank
   (exception 3.)
 2.The key must be unique, which is garanteed in the online by
   a special scheme of 'faked event numbers'
 3.The word 6 of the HEAD bank keeps control on the run type.
   This implies that ALL records (also non-event data) must have
   a HEAD bank. This is already true for online but not for off-line
   (=> change in generators)

NOTE
----
To avoid unnecessary complications it seems to me  clear  that  records
belonging to different runs should be kept well separated  during  data
taking. That is, all event and non_event records of RUN n  should  have
left the L4 farm and send to the IBM before the first record of RUN n+1
is send to the IBM. A. Campbell told  me  that  this  is  garanteed  at
present.
```

```
-------------- end of decription of H1 convention --------------------
```

A.4.1 The standard MAIN program for event processing

```
On the DESY IBM you can list a complete JCL member by putting a *  into
col 1 of the next line:

LIST 'HERA01.H1.FPACK.S(#EVPROC)'

A model for the standard main program and data cards for event-by-event
processing is given below.


************************************************************************
*                                                                      *
*     Standard BOS MAIN program for FPACK data                         *
*                                                                      *
************************************************************************
*                                                     see Remark 1
      PARAMETER (NBCS=600000,NHCS=5000)
      REAL         RW(NBCS)
      COMMON /BCS/ IW(NBCS)
      EQUIVALENCE (IW(1),RW(1))
      COMMON /HCS/ KW(NHCS)
*
*     next statement to allow C/FORTRAN at DESY
      CALL CDUMMY
*
*     Initialize BOS
      CALL BNAMES(2000)
      CALL BOS(IW,NBCS)
      CALL BOS(KW,NHCS)
*                                                     see Remark 2
*     read data cards
      CALL FPARMR(5)
*                                                     see Remark 3
*     read data cards
      CALL BREADC
*                                                     see Remark 4
*     open database
      JOBNR = MDB('/JOB')
*                                                     see Remark 5
*     event processing loop
   10 CALL FSEQR('BOSINPUT',IRET)
      IF(IRET.LT.0) GOTO 100
*                                                     see Remark 6
*     call modules here
```

```
C     CALL ...
*                                                    see Remark 5'
*     flag event eventually for output
C     CALL FSEQW('BOSOUTPUT')
      GOTO 10
*                                                    see Remark 4'
  100 IND = MDB('/END')
      CALL FPARM('CLOSE')
*------------------------------------------------------------------*
*
*     Remark 1: these statements declare the  size  of  the  basic  BOS
*     common /BCS/, of the secondary common /HCS/, which  is  used  for
*     BOS histograms,  and  initialize  the  BOS  system  for  the  two
*     commons. The number of different bank names  allowed  is  set  to
*     2000. The necessary length of the basic  common  depends  on  the
*     modules  called.  The  number  of  600000  words   (2.4   Mbytes)
*     represents a rather large  number,  probably  necessary  if  many
*     reconstruction modules are called. In the BOS  printout  of  your
*     job you can find information on the space really used.
*
*     Remark 2: the subroutine FPARMR reads  from  the  standard  input
*     unit 5 the statements to OPEN the  input  and  eventually  output
*     file(s),  and  optionally  the  RSELECT  cards  for  the   record
*     selection. The  last  of  these  statements  should  be:  END  to
*     separate them from the input to BREADC.
*
*     Remark 3: the subroutine BREADC called here reads from the system
*     input unit text-lines until a text-line with: ENDQ The text-lines
*     are interpreted, the content is stored as BOS banks and available
*     within the program. This option is used by many applications.
*
*     Remark 4: by a call to the H1 data base  function  MDB  with  the
*     argument '/JOB' the H1 data base is opened  within  the  job  for
*     read access. This call is necessary for  any  further  data  base
*     access (for example from some reconstruction  module).  The  OPEN
*     for the data set is done internally.  The  returned  value  is  a
*     unique job_number, which is available from some  common  too.  At
*     the end of the job (just before STOP) one should call  MDB  again
*     with the argument '/END'. The call  should  always  be  done,  to
*     close the data-base file correctly, and is essential, if the  job
*     is adding new data to the data base.
*
*     Remark 5: within the event processing  loop  the  call  to  FSEQR
*     makes the transition from one event to the  next  one.  In  FSEQR
*     first the previous event is  written,  if  output  was  requested
*     (FSEQW, see below). Then the previous event is  dropped  and  all
*     banks no longer needed are removed. The next event  is  read  and
*     then FSEQR returns. The input  file  is  given  by  the  symbolic
*     file_name  (first  argument);  it  is  recommended,  to  use  the
*     symbolic file name 'BOSINPUT'. The  second  argument  IRET  is  a
*     return flag; the value is +1 if an event has been  read.  At  the
```

```
*     end-of-data condition, the  return  flag  has  the  value  0  and
*     usually a final loop through the modules is done (see remark  6).
*     At the next return the value of the flag is -1, and  the  program
*     should stop (IF(IRET.LT.0) GOTO 100).
*       FSEQR is assuming: standard array IW/RW in /BCS/
*                          list E for input and output of banks
*                          list R to delete banks after E list output
*                          list S used internally
*       (lists used for modules steering).
*     After end-of-data a RESTART of  the  reading  loop  is  possible,
*     either with another file (DANAME) or with the same file  after  a
*     rewind (CALL FNRWND(DANAME)).
*
*     Output: there may be up to four (4) output streams. In  order  to
*     write the actual banks, the user has  to call  CALL FSEQW(DANAME)
*     with DANAME = symbolic file-name for output. It is recommended to
*     use 'BOSOUTPUT' for the standard output file. Writing is  delayed
*     until the next call to FSEQR and is done before the new record is
*     read in.
*
*     User calls to skip a run or to stop the reading loop:
*       CALL FSEQS     to skip the actual run (if some run is defined)
*       CALL FSEQE     to stop the reading loop
*
*     Classes:
*     An event_record may be assigned to up to 30 classes.  The RSELECT
*     statement allows to select  events  from  certain classes,  which
*     reduces  the  reading  time  considerably. There  are    utility
*     routines to handle the class information.
*     Get class information for current event in local array:
*       INTEGER NCLASS(30)        declare local array
*       CALL GETCLA(NCLASS)       get data in to local array
*     After this call then local array is filled, with
*       NCLASS(I) = 0             event not assigned to class I
*       NCLASS(I) = 1             event assigned to class I
*     The  local  array  may  be  modified  by  the  user;  the   user
*     modification is stored with the event by the call
*       CALL SETCLA(NCLASS)
*     The current event can be assigned to a certain class, for example
*     class 17, by the following code:
*       INTEGER NCLASS(30)
*       ...
*       CALL GETCLA(NCLASS)
*       NCLASS(17)=1
*       CALL SETCLA(NCLASS)
*
*     Alternatively the current event can  be  assigned  to  a  certain
*     class ICLASS by the utility routine SETCLI:
*       CALL SETCLI(ICLASS)       1 <= ICLASS <= 30
*     For example the assigment to class 17 can be done instead by  the
*     four statement above by only one statement:
```

```
*      CALL SETCLI(17)
*
*     Remark 6: all standard event generation, detector simulation  and
*     reconstruction programs are in the form of modules. Modules  have
*     to be called in the right order and should follow  a  few  rules,
*     but otherwise there are no restrictions. It is recommended to  do
*     physics analysis in the same style. If you have written a  module
*     with some non-trivial output, it can be considered as a candidate
*     for a general H1 library.
*
*----------------------------------------------------------------------*
      STOP
      END
```

```
//* The next two lines are explained below under: OPEN
//*G.FT01F001 DD UNIT=(CART,,DEFER),DISP=              input file
//*G.FT02F001 DD UNIT=(CART,,DEFER),DISP=              output file
//G.SYSIN DD *
*----------------------------------------------------------------------*
*                                                                      *
*       Short HELP for statements, read by FPARMR(5) ...              *
*                                          until ...  END             *
*        (for complete information see FPACK manual)                  *
*                                                                      *
*----------------------------------------------------------------------*
*
*  At present the following keywords are supported:
*
*  Keyword               Function
*  -------------------   ---------------------------------------------
*  OPEN                  Open a file (local or via server on a host)
*  RSELECT               Define selection criteria for records on file
*  REWIND                Rewind a file
*  CLOSE                 Close a file
*  FILECOPY              Copy a file
*
*  IOSTATISTICS          Print current IO statistics
*  PRINT       p_level   Set print level to p_level
*                        (by default =1 in batch, =0 interactive)
*  TIMESTOP     s_time   Define time for stop before expiration of
*                        jobstep time (use s_time > 1.05 sec at DESY)
*  END                   Last card (FPARMR returns to calling pgm)
*
*  Rules for text:
*  Statements are in free form. Keywords may be in upper or lower case,
*  but text, either within single or double quotes ('text' or  "text"),
*  is taken as it  is  written.  Commas  between  different  items  are
*  allowed, but not necessary. A ! signifies start of a  comment  in  a
*  line. A & as last character (before a comment) indicates,  that  the
```

```
*  statement is continued in the next line. Do not forget the &, if the
*  statement is continued!
*
*  Pure comment lines are ...
*       lines with a * in column 1,
*       blank lines, and
*       lines, where the first nonblank character is a !,
*  and are ignored (even within continuation lines).
*
*-----------------------------------------------------------------------
   TIMESTOP 1.05             ! set time parameter
   PRINT    3                ! set print level
*-----------------------------------------------------------------------
*
* Input file
*
   OPEN BOSINPUT UNIT=1 &
   File="H01GRE.LEPTO.NC.DEEP.SCAN5.03SIM204"
*  If  the  file  is  on  a  cartridge,  you  have  to   activate   the
*  corresponding DD card (see above) by removing  the  *  in  the  line
*  //G.FT01F001 ... and you have to add the keyword DEFER to  the  OPEN
*  statement.
*
* Output file (if not existing, write-calls in FPACK are ignored)
*
*  OPEN BOSOUTPUT UNIT=2                       &
*  File="F14BLO.FPACK.TEST03",                 &
*  Action=readwrite, status=new, recl =23400,  &
*  nrec=100, nrec2=50
*  If  the  file  is  on  a  cartridge,  you  have  to   activate   the
*  corresponding DD card (see above) by removing  the  *  in  the  line
*  //G.FT02F001 ... and you have to add the keyword DEFER to  the  OPEN
*  statement.
*
* Selection of records (banks) with the RSELECT statement
*  RSELECT BOSINPUT                  &   ! selection on input
*  RECORD = 1 3:4 6 8 11:50          &   ! records numbers
*  CLASS  = 1 2 7:10                 &   ! record classes
*  RECNAME = 'RUNEVENT'              &   ! record names
*  RECRANGE = 'RUNEVENT' 1:10        &   ! record ranges in run #'s
*  NOTBLOCK  = 'RUNEVENT' 'HEAD'     &   ! blocks within record
*  STOPMB    = 10.0                      ! total space in megabyte
END
```

A.4.2 A mini HELP for BOS

```
The BOS common
--------------
```

Data are stored in banks in the array IW (for integers) / RW (for floating point numbers) in the common /BCS/ (the use of other commons, which is possible, is not recommended). The declaration in a subroutine is:

```
      COMMON/BCS/  IW(1000)
      REAL         RW(1000)
      EQUIVALENCE (RW(1),IW(1))
```

with a fixed (dummy) length, the actual length of the common is defined in the MAIN program.

Named banks

Data are stored in so called banks, which have a certain length and represent a piece of contiguous storage somewhere in the BOS common. Named banks have a 4-character name, for example 'TRAC', and a number (any integer). The index of a certain bank is determined and returned by the function NLINK, for example

```
   IND = NLINK('TRAC',0)
```

returns in IND the index of the bank ('TRAC',0), or zero, if the bank does not exist

Tables and banks

The H1 collaboration uses a certain convention to store data of tabular form in banks. A table has a certain fixed number of columns, NCOL, and a variable number of rows, NROW. An example are tracks, where the parameters of one track are stored in one row. By convention the two dimension parameters of the table, NCOL and NROW, are stored in word 1 and 2 of the bank, and are followed by the data words of the first row, the second row etc.

```
        word 1: NCOL = number of columns
        word 2: NROW = number of rows

        word 3:        column 1 of row 1     )
        word 4:        column 2 of row 1     )   row 1
        ...            ................      )
        word 2+  NCOL: last column of row 1  )

        word 3+  NCOL: column 1 of row 2     )
        word 4+  NCOL: column 2 of row 2     )   row 2
        ....           ................      )
        word 2+2*NCOL: last column of row 2  )
```

Chapter B

B.1 Basic IO subroutines

In this chapter the basic subroutines for all operations with F-files are described. The first action for a file has to be the opening of the file, and this has to be made before any call of input or output subprograms. The standard way is as described in chapter A.3; with the OPEN statement the actual (physical) opening is done, and the characteristics of the file is made known to the FPACK system. The user may also do the actual (physical) opening directly by other means. Then the file characteristic has to be made known by the OPEN statement with keyword NOOPEN, or the user has to call subroutine FUPARM (explained in appendix D).

Input/output in packages like BOS or LOOK is done with one subprogram call, as described within chapter A. In this chapter the basic calls for input and output are explained. Before any call of input or output routines, the file has to be selected, either by specifying the unit number directly, or by specifying the symbolic file name, the dataname. The selection is done separately for input ("R" for Read) and for output ("W" for Write) and remains valid, until it is changed. A selected file can be switched off by selecting the unit number zero.

Input and output of an F-record is done by a series of calls. The first call is relevant for the RECORD KEY, with the information (see chapter A.1. General introduction):

Record name	string of 8 characters	RECNAME
Record number A	integer	NUMRA
Record number B	integer	NUMRB
Classification	integer	ICL

This call is followed by calls for so called data blocks with the information:

Name of data block	string of 8 characters	NAMEDB
Number of data block	integer	NUMDB
number of columns	integer	NCOL
number of rows	integer	NROW
block format	string of characters	FORMAT

A data block has a name (string of up to eight characters) and a number (any integer). The data block can be considered as a table with NCOL columns and NROW rows, and these two parameters NCOL and NROW are stored together with the name and number of the data block in a data block header (of course NCOL or NROW can have the value 1). In addition there is a format description of the data block in a text string of variable length; this format definition is similar to a FORTRAN format statement information and is described in subchapter B.1.3.

The data block itself is transmitted by another call between the internal F-record buffer and an user array. The length of the data block is usually equal to the product NCOL*NROW. In some applications the length of the data block may not yet be known at the time the header is added to the F-record. In other applications the length of the data block may be larger than any user array. Therefore, in FPACK, it is (1) not required, that the length of the data block is equal to NCOL*NROW, and (2) the data of the data block may be transmitted in several parts between the user array and the F-record buffer (the size of the parts depending on the size of the user array). Thus the data transmission is independent of the values of the parameters NCOL and NROW and these may be considered as comments.

A logo for FPACK, giving the version number, is printed by

```
   CALL FLOGO
```

B.1.1 Output

The file has to be selected first, either by specifying the unit number directly (FWUNIT) or by specifying the dataname (FWNAME). The attributes of the file have of course to be defined before by a call of subroutine FPARM or FPARMR. All following calls of other subroutines for writing use the unit defined by this call, until the unit is changed by another call. The argument UNIT may be zero to switch off the last used unit number.

```
                 ---
  CALL FWUNIT(LUN)        select the unit (1...99) for writing
or               --------
  CALL FWNAME(DATANAME)   select the symbolic file name for writing
```

Writing a record: the content of a logical record is defined by a sequence of calls, the first one (after definition of the unit) is the call to define the complete key of the record.

```
             ------- ----- ----- ---
  CALL FWKEY(RECNAME,NUMRA,NUMRB,ICL)        define record key
```

followed by any number of pairs of calls:

```
            ------ ----- ---- ---- ------
  CALL FWHDR(NAMEDB,NUMDB,NCOL,NROW,FORMAT)   define data block header

     where NCOL = number of columns
           NROW = number of rows
                  ( NCOL * NROW has to be equal to the number of words
                    written using FWDAT calls )
```

```
                 -- -----
     CALL FWDAT(NR,ARRAY)          output of array ARRAY(1)...ARRAY(NR)
```

Several calls of FWDAT are allowed under the same data header; the data block will contain all data given in the order of calls.

A logical record is finished by the call:

```
  CALL FWEND(IERR)   end of user record
             ----
```

where IERR = 0 indicates "no error".

An F-file is finished by the call:

```
  CALL FWEOD   end-of-data (writing of the buffer)
```

F-records are blocked internally using a buffer. After finishing one logical record usually the buffer contains still some data from the logical record. By the call of FWEOD the buffer is written, and this is the reason for having this call. Writing to the same file can continue after a call of FWEOD. If a (write) unit is closed before FWEOD is called by a call to FPACK closing call FCLOS or is rewound by a call to the FPACK rewind routine FRWND, the last buffer (if existing) will be written automatically.

Some options may be changed for the currently selected output unit by

```
              ------ ------
   CALL FWPARM(OPTION,IVALUE)
```

```
OPTION = 'ORIGIN'  IVALUE = 0 ... 15 (1=BOS, 2=LOOK, ...)
OPTION = 'TEXTCOL' IVALUE = 0 ... 20 (column structure for text output)
OPTION = 'NOCONVERT' IVALUE = not used
OPTION = 'CONVERT'   IVALUE = not used
```

The first call adds another flag to the record header, that indicates the origin of the data (either BOS or LOOK or ...). The second call gives a column structure to text files, which uses more space, but simplifies editing of data. The third option sets a flag to inhibit conversion.

Output to KEYED ACCESS and ORDERED ACCESS files

Keyed and ordered access files have special properties and allow certain additional operations:

(1) the keys of records have to be unique (there can be no two records on the file with the same key), (see DUPLICATE KEY condition below)
(2) records with a certain key can be replaced, and
(3) records with a certain key can be deleted.

These features are essential for data base applications.

DUPLICATE KEY:
If a record key is specified in a write operation (FWKEY), which is identical to the key of an existing record on the file, then the old record is automatically deleted. Thus overwriting (updating) is done in the case of records with duplicate keys!

The existence of a record with a certain key can be checked BEFORE a write operation by the logical function FQDUPL:

```
    LOGICAL FQDUPL, LL
                 -------- ------- ----- -----
    LL = FQDUPL(DATANAME,RECNAME,NUMRA,NUMRB)    TRUE. if record exists
```

The function will return the value .FALSE., if the record does not exist on the file with data name DATANAME or the file is not existing or the file is not keyed access. The function does NOT make any change on the file and can be used to test the existence of a record on a keyed or direct access file at any time.

A record with a certain key is DELETED (if existing) by the call:

```
               -------- ------- ----- -----
    CALL FWDEL(DATANAME,RECNAME,NUMRA,NUMRB,IERR)        delete record
                                            ----
    IERR = 0   if deleted (keyed or ordered access)
```

```
            = 1   if not existing or no keyed/ordered access file
```

A VERY IMPORTANT NOTE: If you are writing to an ordered access file, you must close it explicitly, using the CLOSE statement, before the end of your job. Without it all the modifications of the file will be lost, and file will remain the same, as before the job started.

Remember, that "direct" access mode is used for ordered access files. So, on some machines (e.g. IBM mainframes), the size of the file, (parameter NREC in an OPEN statement) cannot be changed after the file creation. You must code large enough value for NREC parameter to assure that the file can keep all your data.

If you do some replacements or deletions in an ordered access files, then the space occupied by old records is utilized only at file closure time, replacing records will be written to free parts of the file or appended to its end. However, it doesn't mean, that file will grow up infinitely. If you replace one record 1000 times, in one or several jobs, then additional space is needed only for one record, not for 1000 of them. On some machines (e.g. IBM mainframes), file cannot be expanded during writing, so it must have enough free space, to keep all the replacing records. If you want to replace all the records in an ordered access file, it must have twice the capacity, to keep all your data. However, if it has twice the capacity, you can replace all the records any number of times in one or several jobs.

B.1.2 Input

The file has to be selected first, either by specifying the unit number directly (FRUNIT) or by specifying the dataname (FRNAME). All following calls of other subroutines for reading use the unit defined by this call, until the unit is changed by another call. The argument UNIT may be zero to switch off the last used unit number.

```
               ---
  CALL FRUNIT(LUN)       select the unit (1...99) for read
or             --------
  CALL FRNAME(DATANAME)   select the dataname for read
```

Reading a record: the first call (after definition of the unit) is the call to read the complete key of the next record.

```
  CALL FRKEY(RECNAME,NUMRA,NUMRB,ICL,IERR)     get RECORD KEY
             ------- ----- ----- --- ----
```

```
           =  0   no error
      IERR = -1   end-of-data
           >  0   error code
```

There is an alternative call FRKEY2 (see end of this subchapter).

The call FRKEY (or FRKEY2) can then (for IERR=0) be followed by pairs of calls:

```
  CALL FRHDR(NAMEDB,NUMDB,NCOL,NROW,FORMAT,NCH,IERR)  get data header
             ------ ----- ---- ---- ------ --- ----       of next block



     where NCOL          = number of columns
           NROW          = number of rows
           FORMAT(1:NCH) = format code
                         >  0 error code
           IERR          = -1 end-of-data for the current event
                         =  0 no error


                         ----
     CALL FRDAT(NR,ARRAY,NDIM)     get data into ARRAY(1)...ARRAY(NR),
                -- -----           which has dimension NDIM
```

If the returned value of NR is equal to NDIM there may be more data, which may be obtained by subsequent calls. For NR=0 no more data are available for the current data block.

The FRKEY call will return the key of the next record. By subsequent FRKEY calls it is possible to search sequentially for certain keys, and to read only data blocks of selected records. Similarly, it is possible to read a subset of the data blocks of a record: the FRHDR call will return the header information of the next block of the same record. The return flag IERR of the FRHDR call will have the value -1 after the last data block header of the record.

Alternative to FRKEY: a call to FRKEY provides the key of the next record (to be read with calls FRHDR, FRDAT ...). In some applications the user may want to read through a file using FRKEY, until a certain record key has been found. Then the complete record has to be read in, may be by a subroutine like FRBOS or FRLOOK. If such a subroutine would start with a call of FRKEY, the next record key would be returned, thus "skipping" the requested record. There is an alternative FRKEY2 to FRKEY, which has the identical calling sequence as FRKEY: FRKEY2 returns the same key information (no new reading) as a previous call to FRKEY, if no FRHDR or FRDAT call is done after the call to FRKEY. Otherwise the functions of FRKEY2 and FRKEY are identical. Within FRBOS and FRLOOK the alternative FRKEY2 is called. This allows to make a search for a certain record outside these subroutines with FRKEY.

Input from KEYED ACCESS or ORDERED ACESS files

Input of records from keyed or ordered access files has additional options and special properties concerning the order of records:

(1) the order of records read is the order of the keys, not the order of writing,

(2) it is possible to position the file for reading to a certain key with default starting position at the smallest key, and

(3) forward and backward reading is possible (default is forward).

Thus the records are automatically sorted. The features allow certain data base operations.

Increasing order of keys is defined by the order of the record name and the order of the record numbers. . The order of record names is in the collating sequence of the characters in ASCII code, which for letters should be the usual alphabetical order. For example the order of record names would be

```
   ALIBABA  BRIDGE  ...  ZOOMING
```

as you would probably expect. Within records with the same record name, the order is given by the numerical value of the tow record numbers, and here you may be surprised by the order of records with negative numerical values. Increasing is the order of values

```
             0
             small positive values
             large positive values
             large negative values
             small negative values
             -1
```

This order looks funny, but has some advantages: starting reading forward with 0 will give you all records of the same title, starting backward from -1 will give you too all records of the same record name (in reverse order). Reading forward from -1 will give you the first record of the next record name.

The following call(s) define the new position of an input file and the new direction (after having of course defined the unit/data name for input before)

```
            ------- ----- -----
   CALL FRPOSF(RECNAME,NUMRA,NUMRB,IERR)    direction forward
                                    ----

            ------- ----- -----
   CALL FRPOSB(RECNAME,NUMRA,NUMRB,IERR)    direction backward
                                    ----
```

```
   where IERR = 0  for existing keyed access file
              = 1  if file not existing or not keyed access
```

The next input call (FRKEY) will give the record with the key given in the argument, if existing, otherwise the next or previous one, depending on the direction. In subsequent calls it will give the next/previous record, keeping the requested order. The end-of-data condition is given, as usual, by the argument of FRKEY at return. Note that this condition is valid after the last record for forward reading and after the first record (!) for backward reading.
The sequence of records in reading is not disturbed by intermediate write operations (this is a propertiy of FPACK, not of the access method).

Special case is :

```
   CALL FRPOSF('MINIMKEY',NUMRA,NUMRB,IERR) direction forward from
                                            the beginning of the file

   CALL FRPOSB('MAXIMKEY',NUMRA,NUMRB,IERR) direction backward from
                                            the end of the file
```

B.1.3 Data block formats

The format of the block (see Table 1) describes the type of the data stored in the block and is used for any conversion between different machines. The format can be either

- mixed format (integer, floating point, text), or
- bit-packed format (8- or 16- or 32-bit data).

Mixed format:
The format description follows the same rules as FORTRAN format statements, except that no length information is specified:

```
      I   integer
      F   floating point
      A   text (hollerith, 4 characters per word)
```

The character string defining the format is of the type

```
            'f1, f2, ...',
```

where the fi are format codes nI, nF, nA or n(...) for a group format specification with constant n (missing n means 1), for example

```
            '2I,(5F,2(I,3F,A))'
```

According to the FORTRAN rules, when the format control reaches the last (outer) right parenthesis and there are data words left, the format starts again by the last preceding right parenthesis, including its group repeat count, if any, or, if no group specification exists, then at the first left parenthesis of the format specification.

Special formats:
The 32 rightmost bits of each machine word are considered. These 32 bits may contain either 32-bit information or two 16-bit informations or four 8-bit informations. The format description is

'B32'   for 32-bit information
'B16'   for 16-bit information (two per word)
'B08'   for 08-bit information (four per word)

and is valid for a whole data block. No conversion is done for these data, but on certain machines a reordering has to be done.

I	integer
F	floating point
A	text (4 character/word, Hollerith data in integers
B32	32-bit data (right-adjsuted in a word)
B16	16-bit data (two per word, right adjusted in a word
B08	8-bit data (four per word, right adjusted in a word

Table 1. Format information for mixed format (above) and for special formats (below); the latter one is valid for a whole data block.

It is recommended to use only characters from ASCII 7 bit standard (128-character set, see Table 2) or the corresponding subset of EBCDIC characters. The remaining special characters may not be converted correctly during the transfer between computers with different architecture.

```
+-----------------------------------------------------+
|                          1         2         3    |
|                0123456789012345678901234567890 1  |
+--------------+----------+----------+----------+---+
|   0 ...  31  |  ........system codes...........   |
|  32 ...  63  |   !"#$%&'()*+,-./0123456789:;<=>?  |
|  64 ...  95  |  @ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_  |
|  96 ... 127  |  `abcdefghijklmnopqrstuvwxyz{|}~   |
+-----------------------------------------------------+
```

Table 2. Standard ASCII 7 bit character set (printed characters depend on the printer font)

B.1.4 Further file operations

The file operations described here apply to all applications of FPACK, i.e. the operations can be done with BOS or LOOK or other data in F-files. Arguments are either the unit number or the symbolic file name, as specified in FPARM.

Rewind a F-file:

```
                ---
   CALL FURWND(LUN)
or                             rewind file
                --------
   CALL FNRWND(DATANAME)
```

If the status of the file at the time of the call is "writing", the last buffer (if existing) is written before the rewind operation.

Close a F-file:

```
                ---
   CALL FUCLOS(LUN)

or                             close file
                --------
   CALL FNCLOS(DATANAME)
```

If the status of the file at the time of the call is "writing", the last buffer (if existing) is written before the close operation. The unit number and data name can be used afterwards for another file.

Close ALL F-files:

```
   CALL FCLOS                  close all files
```

All F-files open at the time of the call are closed (buffers, if existing, are written before the close operation).

Get current record key information:

```
   CALL FRQKEY(RECNAME,NUMRA,NUMRB,ICL,IORIG,IRECNC,INEXT)
               ------- ----- ----- --- ----- ------ -----
```

```
      If INEXT=0: information is from record, already read,
      if INEXT=1: information after FRKEY call (that means: FRKEY
                  has been called already)
      IORIG  = origin code (1=BOS, 2=LOOK, ...)
      IRECNC = record number * 100 + position within record
```

Print current record key information

```
   CALL FPRKEY
```

Printing of a table of content:

```
  [CALL FNRWND(DATANAME)]   eventually rewind before
               --------
   CALL FRTABC(DATANAME)    read file until end-of-data and print
                            table of content
```

Note, that reading starts at the actual position and continues until the end-of-data is reached. A call of FNRWND should be given before, if the complete table is required. At return the position is at the end-of-file.

Copying records from one FPACK file to another FPACK file:

```
              ------ ------- ----
   CALL FCOPY(DATAIN,DATAOUT,NREC,IFLG)    copy NREC logical records
                                 ----     if NREC=0, copy all
```

The symbolic file name of the input file is DATAIN, of the output DATAOUT. NREC logical records are copied from DATAIN to DATAOUT, the last buffer is written before the return. If NREC=0, all logical records are copied. IFLG contains number of errors.

B.2 Record classification, index and ordered files

The purpose of the record classification is to allow for a fast access to a subset of data.

B.2.1 Record classification

A record from a file can be assigned to one or to several classes. Classes are numbered by

```
   ICLASS = 1 ... 30,
```

thus there is a maximum of 30 classes. Internally the class information is stored in one word ICL, with each class represented by one bit. A certain class may represent a certain physics channel, for example in an e+e- experiment class 1 may be assigned to Bhabha events, and class 2 to muon pairs; if an event is a candidate for both classes, then the classword will contain flags for classes 1 and 2. In an analysis for muon pairs the user can specify, that only record, assigned to class 2 are read.

Although the assigment of classes is arbitrary, it is recommended for physics experiments to divide the classes into three parts, according to the scheme:

ICLASS	classification
1 ... 10	online (trigger, filter)
11 ... 20	offline
21 ... 30	private analysis

The classification word ICL is part of the data record key. It is available by the FRKEY call, where the record key of the next record is returned.

For the simple access to the classification by the user there are subroutines for packing and unpacking of the classword ICL and an array NCLASS(30), to be declared as

```
    INTEGER NCLASS(30)
```

The calls are:

```
             ---
   CALL FCLUP(ICL,NCLASS)      unpack class word into array
                 ------




                 ------
   CALL FCLPA(ICL,NCLASS)      pack classword from array
             ---
```

The content of the array is interpreted as follows:

```
   NCLASS(ICLASS) = 0    record not assigned to class ICLASS
   NCLASS(ICLASS) = 1    record assigned to class ICLASS (any nonzero
                         value is interpreted as 1)
```

While the two packing/unpacking routines above affect only arguments of the routines and have no further effect, there are additional routines to edit the classification of records. Editing means, that

- the classword ICL can be reset,
- the classword ICL can be defined, and
- class assignments can be added to the classword ICL.

The two routines for updating the classword have as first argument the symbolic file name of the file, corresponding to the record, which is classified.

```
                -------- ------
CALL FSCLAI(DATANAME,ICLASS)   set class ICLASS in the classword
                               (logical OR of class bit)

                -------- ---
CALL FSCLAS(DATANAME,ICL)      set class word to ICL
```

The present value of the classword is obtained by the call

```
                --------
CALL FGCLAS(DATANAME,ICL)      get class word ICL
                         ---
```

The effect of these calls depend on the type of file, specified in the calls (data or directory file) and on the action attribute (read or write) of the file. The different cases are discussed later.

B.2.2 Internal file index in an ordered access file

Each ordered access file contains an internal index, which contains the information on record keys in the file including the classword plus information on the position of the record within the file. The purpose of the index is to make the access to a subset of records on the data file fast. The use of an index for record reading represents a negligible overhead, thus there is no disadvantage in using the index even for reading the complete file.

Below the use of index for READING is explained (there are several cases possible for writing, these are discussed in the chapter B.2.3)

The use of index for reading is very simple. No new additional subroutine call is necessary for reading. As input file the user has to specify the ordered access file and code ORDERED in an OPEN statement.

The record key including classification is read from the index (in the FRKEY call). For following calls to the data access calls FRHDR and FRDAT the record itself is read. The position of the data record(s) is taken from the index. The "direct" access mode is used to get the records. The record key, returned at the call of FRKEY, is taken from the index. If the user then decides to read the data (by calling FRHDR and FRDAT), the data record(s) are read.

B.2.3 Update of class bits and writing

Several cases have to be distinguished in the actions after updating the classword for a record. The subroutines FGCLAS, FSCLAI and FSCLAS are already described in chapter B.2.1. They can be used to retrieve (FGCLAS) and update (FSCLAI, FSCLAS) the present value of the classword. Several of these calls are possible, for example to set several classes.

Case DATANAME is a data file, being read

The classword updated is the classword of the last record (key) read. The update of the classword is done immediately and a subsequent call of FRQKEY will give the updated value of the classword.

Case DATANAME is a data file, being written

The classword updated is the classword of the next record (key), to be written. The record with the updated classword is written with the usual calls for writing data records.

B.2.4 Index files

Index files

FPACK allows the creation of so called "index" files containing only record keys and references to other files, where real data records reside. Index files can only be used for reading the real data (not for writing it). The index file has to be opened as an ordinary sequential FPACK file. No special keywords or modes are needed. Switching between an index file (to get the record key) and data files (to get the data) is done automatically and hidden for the users. In some sense index file can be considered as a "directory of events" allowing fast access to the selected events. All the searches and selections are made using the information from index file only so data file may be not opened at all.

Index file structure

As mentioned above, the structure of index file is similar the structure of sequential binary FPACK file. It consists of the logical records each of them, in turn, consists of record key and data blocks. A record key is used to distinguish between real data and index files. The record name of all record keys in the index file is FPINDEXK and each record describes one real data file. The data blocks contain the data file name, selected record keys of data file and, optionally, the host name (in case of a remote data file), volume name (for cartridges) and record length. When FPACK finds a record with the name FPINDEXK, it reads the data file description from this record. It hides the record FPINDEXK from the user and gives him the record keys found in this record in the index file. If user wants to read the real data (using FRHDR/FRDAT) then the data file is opened and data are extracted from it. The way of opening the data file depends on a media where data file resides. For disk and staged data files FPACK uses direct access. FPACK keeps in mind the origin of data file (cartridge, stage or disk) and if you don't say to FPACK something else (DDEFER or DSTAGE), it will try to work with data file according to its origin. For example, if you have index to staged data file, but, in that time your file is not in a staged state, FPACK will try first to stage this file. If it's failed then it will call for cartridge data file directly.

IMPORTANT: The index file must not exist before its creation.

Creating index files

There are three ways to create index files. The simplest one is to create them in a special job using the subroutine FPBLDI. This way doesn't allow you to use FPACK select commands, so FPACK builds "full" index of data file(s).

```
                ----- ------
   CALL FPBLDI(NCHAN,FNINDX)                               build index file
```

- NCHAN - logical unit number of the list of data_files. The most frequently used unit number is 5, the standard input unit. If the user wishes to use a different unit, he has to open the file containing the list of data_files himself.
- FNINDX - file_name of the new index file (must not exist before)

On the DESY IBM you can list a complete JCL member with an example for index file creation by putting a * into col 1 of the next line:

```
LIST 'HERA01.H1.FPACK.S(#MK_IDX1)'
```

The second way is decribed in the chapter A.3.6. The IDXCOPY statement. On the DESY IBM you can list a complete JCL member with an example for index file creation by putting a * into col 1 of the next line:

LIST 'HERA01.H1.FPACK.S(#IDXCOPY)'

The third possibility is to call routines FPCURX, FPCLIX or FPWRIX. They add one entry to the index file. The entry consists of a real data file_name and one record key. FROM and TO must be opened FPACK files. FROM must not be opened with NOOPEN option, but DEFER or STAGE options are allowed.

FPCLIX and FPWRIX are especially useful, if you want to change the classification word and enter your own classification of events, for example, after your own physical analysis. Such classification is kept only in your private index file and may be used in the following selections when it will be read.

```
            ---- --
     FPCURX(FROM,TO,IERR) - write the current logical record
                    ----    information into the index file



                            FROM - logical name of the data file
                            TO   - logical name of the index file
            ---- -- ---
     FPCLIX(FROM,TO,ICL,IERR) - write the current logical record
                        ----    information into the index file and


                                change the classification word ICL
                                IF(ICL.EQ.-1) NO CHANGE
```

If the current record has exactly the given record key, then the reading position in FROM is not changed by FPWRIX. In any other case FROM is scanned to locate the needed record. Scanning is done from the current record to the end of file. If this was unsuccessful, the file is rewound and another scan is performed. If it fails too, error is reported.

```
            ---- -- ------- ----- ----- ---
     FPWRIX(FROM,TO,RECNAME,NUMRA,NUMRB,ICL,IERR) - write one entry
                                            ----    into the index file
```

On the DESY IBM you can list a complete JCL member with an example for adding information into an index file by putting a * into col 1 of the next line:

```
LIST 'HERA01.H1.FPACK.S(#MK_IDX2)'
```

B.3 Input/output of BOS data

There is one standard event loop routine for reading and writing of event BOS data, called FSEQR, which has a subroutine similar to the subroutine BSEQR. Subroutine FSEQR reads and writes F-files only. For the transition period from the old BOS files to F-files there is a second version BFSEQR, which allows in addition to read old BOS files. In addition there are subroutines for single record input/output.

```
Subroutines FSEQR and BFSEQR
----------------------------
```

Subroutine FSEQR has a functionality similar to the BOS subroutine BSEQR and can be used for the input/output of event records in the standard event processing loop.

```
                --------
  10 CALL FSEQR(DATANAME,IRET)
                         ----
     IF(IRET.LT.0) GOTO ... (end-of-data)
     ... (call modules)
     GOTO 10
or               --------
  10 CALL BFSEQR(DATANAME,IRET)
                          ----
     IF(IRET.LT.0) GOTO ... (end-of-data)
     ... (call modules)
     GOTO 10
```

To read old data sets (FORT or EPIO) from unit 1 use BFSEQR with a call CALL BFSEQR('OLDBOS',IRET) (the latter call is ignored in FSEQR). For compatibility with old bos files also the following call is possible:

```
                  ------ ------
        CALL FSEQP(LIMSEC,LIMREC)    with default (2,0)
```

Reading with FSEQR/BFSEQR/FRBOS will stop reading LIMSEC seconds before the time limit or after LIMREC records, where LIMREC=0 maens no limit.

Output is always done by FPACK.

```
FSEQR/BFSEQR is assuming: standard array IW/RW in /BCS/
              list E for input and output of banks
              list R to delete banks after E list output
              list S used internally
(lists used for modules steering, to be documented later).

Arguments:

   DATANAME = symbolic file name of input data set
              (or 'OLDBOS', see above)
        IRET = returned flag with values ...
             =  1   record read
        IRET =  0   end-of-data condition - final loop thru modules
             = -1   after end-of data condition (skip modules)

After end-of-data a RESTART of the reading  loop  is  possible,  either
with another file (DATANAME) or with the same file after  a  rewind  or
close of the data set, see chapter B.1.4).

Writing: There may be up to four (4) output streams. In order to write
the actual banks, the user has to call
                    --------
        CALL FSEQW(DATANAME)

with DATANAME = symbolic file name for output. Writing is delayed until
the next call to FSEQR and is done using the  list  E  before  the  new
record is read in.



Note on keys
------------
For events the record key is defined from old BOS data as
   RECNAME = 'BOSEVENT'
   NUMRA   =  run #
   NUMRB   =  event #
   ICL     =  0
The information is taken from the bank 'HEAD',0, words 2 or 3, (or from
other banks, if these are defined once calls
   CALL FSEQH(name,nr,irun,ievent)
to specify the name and nr of the bank, and  the  indices  of  run  and
event number; CALL FSEQH('HEAD',0,2,3) is assumed by default).

Other records (no bank with run and  event  number  found)  are  called
special, they are counted and the key is defined to be
   RECNAME = 'SPECIAL'
   NUMRA   =  counter
   NUMRB   =  0
   ICL     =  0
```

```
User calls to skip a run or to stop the reading loop etc:
   CALL FSEQS     to skip the actual run (if some run is defined)
   CALL FSEQE     to stop the reading loop
```

Subroutines FRBOS and FWBOS

```
Calls for input and output of sigle records.

Input:          ---
   CALL FRBOS(JW,LUN,LIST,IERR)     read set of banks from unit LUN
              --     ---- ----
   where:
      JW   = BOS array (usually IW)
      LUN  = unit number (integer)
      LIST = list of names
      IERR = returned error flag with meaning
             IERR =  0   no error
                  = -1   end-of-file
                  >  0   read error

Output:
              -- --- ----
   CALL FWBOS(JW,LUN,LIST,IERR)      write set of banks from unit LUN
                          ----
   where:
      JW   = BOS array (usually IW)
      LUN  = unit number (integer)
      LIST = list of names
      IERR = returned error flag with meaning
             IERR = 0   no error
                  > 0   write error

At the end the internal buffer has to be written by the call:
              -- --- ---
   CALL FWBOS(JW,LUN,'0',IERR)       write last buffer
                         ----

Rewind or close for the files can be done with the standard FPACK calls
as described in chapter B.1.4.

The note on keys above applies as well to the subroutine FWBOS.
```

B.4 Input/output of LOOK data

All units must be specified before in a FPARM call.

```
Input:          --------
   CALL FRLOOK(DATANAME,IERR)     read LOOK data
                         ----


   where:
      DATANAME = symbolic file name
      IERR   = returned error flag
               =  0   no error
            IERR = -1   end-of-file
               >  0   read error

Output:         --------
   CALL FWLOOK(DATANAME,IERR)     write set of banks to unit LUN
                         ----     from the current area
                                  (the buffer is written automatically)


   where:
      DATANAME = symbolic file name
      IERR   = returned error flag
            IERR =  0   no error
               >  0   write error
```

At present only the output of a complete area is possible.

Rewind or close for the files can be done with the standard FPACK calls as described in chapter B.1.4.

Note on keys. At present the keys for LOOK records are defined as follows:

```
   RECNAME = area name
   NUMRA   = area number
   NUMRB   = 0
   ICL     = 0
```

B.5 Input/output statistics

The routine prints the input/output statistics for all files connected to FPACK in the moment of calling this routine.

```
   CALL FPSTAT
```

Example:

```
************************************************************************
*                                                                      *
*          I N P U T / O U T P U T     S T A T I S T I C S             *
*                                                                      *
************************************************************************

Data name .......... FPACKIN
File name .......... F99ABC.DATA.INPUT
Host name .......... ---

Unit number ....................................................         1
Access .........................................................  sequential
Form ...........................................................      binary
Action .........................................................        read
Physical record length in bytes ...........................           23400

Number of physical records read ...........................              34
Number of logical records read ............................               6

Number of Kbytes read ......................................            796

Average length of logical record on input .................          130574
Number of data blocks read ................................             531
-------------------------------------------------------------------------
Data name .......... FPACKOUT
File name .......... F99ABC.DATA.OUTPUT
Host name .......... MVS

Unit number ....................................................        11
Access .........................................................  sequential
Form ...........................................................      binary
Action .........................................................       write
Physical record length in bytes ...........................           23400

Number of physical records written ........................              34
Number of logical records written..........................               6

Number of Kbytes written ..................................          795600

Average length of logical record written ..................          130574
Number of data blocks written .............................             531
-------------------------------------------------------------------------
```

B.5.1 Error history report

```
FPACK keeps traceback of errors which occured  during  I/O  operations.
The integer function IFPTRB() returns the depth of the traceback.

To print the information about accumulated errors, it means  the  error
code, the error message and the name of the subroutine,  in  which  the
error has been detected

                      ------
          CALL FPEMSG(IDEPTH)

The integer IDEPTH parameter means the depth of the traceback.
If IDEPTH = -1 the whole traceback will be printed
          =  0 suppress printing
          >  0 prints the traceback of the depth IDEPTH
Any call FPEMSG causes clearing of error trace.

Call FPEMSG will be added in the routine FSEQR in the  nex  release  of
H1UTIL. The recommended usage in user's programs is
          IF(IFPTRB().NE.0) THEN
             CALL FPEMSG(-1)
             GOTO 100
          END IF
```

B.6 File inquire

```
File inquire routine returns several file parameters and input/output
statistics for given symbolic file name.


                --------
   CALL FQFILE(DATANAME,FILENAME,HOSTNAME,LUNPAR,IERR)
                         -------- -------- ------ ----



Input parameter:

      DATANAME*16  = Symbolic data name


Output parameters:

   FILENAME*256 = File name
```

```
HOSTNAME*127 = Host name
LUNPAR(30)   = Array of file parameters
  (1) = Unit number
  (2) = Access  ( 1=seq    2=dir    3=keyed   4=special  5=ordered )
  (3) = Form    ( 1=binary 2=text )
  (4) = Action  ( 1=read   2=write  3=readwrite  4=modify )
  (5) = Physical record length in bytes
  (6) = Number of physical records read
  (7) = Number of physical records written
  (8) = Number of logical records read
  (9) = Number of logical records written
 (10) = Number of Kbytes read

 (11) = Number of Kbytes written
 (12) = Not used
 (13) = Not used
 (14) = Average length of logical records on input
 (15) = Average length of logical records written
 (16) = Number of data blocks read
 (17) = Number of data blocks written
 (18) = Number of data blocks written without format ( B32 added )

 (19 - 30) = not used yet

IERR =  0   no error
     = -1   unknown symbolic file name
```

B.7 Networking

B.7.1 Transferring of binary FPACK files by FTP

```
On IBM/MVS use the command X FTP (not FTP) and do not forget the option
"(REPLACE", if you overwrite an existing IBM/MVS file.

On Apollo use the command FTP. If you transfer a file  from  Apollo  to
IBM/MVS, you can also use a much more convenient 'fftp'.
On Macintosh IIci it is more complicated. You have to start the program
         "BYU_Telnet 2.3 -MacTCP-"
in the index
         "Macintosh HD:NewMacs:Telnet 2.3:Telnet 2.3MacTcp"
(only for Macintosh IIci H1K346 in  1c  #343,  for  other  machines  an
equivalent  directory  must  be  found).  Then  click  on   the   'Open
Connection' under 'File', write the 'Session name' (MVS for IBM/MVS or,
for example, H103 for Apollo), click on  the  field  'FTP-session'  and
click on the field 'OK'. Then you must login  with  the  command  'USER
user_id' and continue in a standard way.
```

Transfer mode MUST ALLWAYS BE set to BINARY during the FTP-session. You have to use the command either

```
   binary                 on  IBM/MVS,  Apollo,  Macintosh IIci, or
   set type binary        on VAX.
```

a12) Transfer from IBM/MVS to Apollo or to Macintosh IIci

It can be triggered on both sides and causes no problems if the IBM/MVS file is properly structured.

a3) Transfer from IBM/MVS to VAX

It is not possible to define properly the file structure on the VAX within a FTP-session. (only in this version of FTP, hopefully). Files created in such a way have 'variable record length'. Transfer into an existing file fails, because record length will be changed. It should be possible to use the CONVERT utility and FDL on the VAX, but there is no experience.

Use the command:

```
       COPY DSYIBM::"[userid]filename/BINARY" vax_file_name
```

to copy an FPACK file to the VAX. The IBM/MVS file can also be directly read record by record from the VAX by specifying the input file name as

```
       DSYIBM::"[userid]filename/BINARY"
```

For record sizes above 4096 you have to use the command

```
       SET RMS_DEFAULT/NETWORK=xx
```

before reading the file. Here xx is the number of 512 Byte blocks needed for the transfer of one record, i.e. xx = 50 is sufficient to read 23400 bytes records.

b1) Transfer from Apollo to IBM/MVS

It requires proper structuring of the IBM/MVS file.

If the IBM/MVS file already exists, its format cannot be changed remotly during the FTP-session. If you trigger the transfer on Apollo, just delete the old file using the FTP command 'delete' if the file lacks proper structure. If you are working on the IBM/MVS, delete the file or allocate it with suitable parameters. (Remember, that if the file is larger then the default size foreseen, then you have to pre--allocate the file to get enough disk space) Then specify the parameters to the IBM/MVS during the FTP-session by typing either

```
 quote "site recfm=f blocksize=### lrecl=###"   ,    on Apollo or

 LOCSITE RECFM=F BLOCKSIZE=### LRECL=###   ,    on IBM/MVS.
```

The ### must be replaced with the physical record length (in bytes) of the file to be transmitted. Note, that this number must be known to the user before he starts the transfer. For Apollo users this inconvenience has been removed by providing a utility 'fftp' which checks the actual physical record length and does almost all for the user.

Examples of possible problems:

If the user does not specify the IBM/MVS file structure during the transfer from Apollo to IBM/MVS, then MVS chooses RECFM=VB, if the IBM/MVS file did not exist, and an attempt to read the transmitted file results in reading errors.
Incorrect specification of the BLOCKSIZE during a transmission from Apollo to IBM/MVS also results in reading errors on IBM/MVS.

b2) Transfer from Apollo to Macintosh IIci

It can be triggered on both sides and causes no problems. No declaration of any special file structure is necessary.

b3) Transfer from Apollo to VAX

It is not possible to define properly the file structure on the VAX within a FTP-session. (only in this version of FTP, hopefully). Files created in such a way have 'variable record length'. Transfer into an existing file fails, because record length will be changed. It should be possible to use the CONVERT utility and FDL on the VAX, but there is no experience.

c1) Transfer from Macintosh IIci to IBM/MVS

It requires proper structuring of the IBM/MVS file.

If the IBM/MVS file already exists, its format cannot be changed remotly during the FTP-session. If you trigger the transfer on Macintosh IIci, just delete the old file using the FTP command 'delete' if the file lacks proper structure. If you are working on the IBM/MVS, delete the file or allocate it with suitable parameters. (Remember, that if the file is larger then the default size foreseen, then you have to pre- -allocate the file to get enough disk space) Then specify the parameters to the IBM/MVS during the FTP-session by typing either

```
 quote site recfm=f blocksize=### lrecl=###  ,    on Macintosh IIci or
```

```
 LOCSITE RECFM=F BLOCKSIZE=### LRECL=###  ,      on IBM/MVS.
```

The ### must be replaced with the physical record length (in bytes) of the file to be transmitted. Note, that this number must be known to the user before he starts the transfer.

c2) Transfer from Macintosh IIci to Apollo

It can be triggered on both sides and causes no problems. No declaration of any special file structure is necessary.

c3) Transfer from Macintosh IIci to VAX

It is not possible to define properly the file structure on the VAX within a FTP-session. (only in this version of FTP, hopefully). Files created in such a way have 'variable record length'. Transfer into an existing file fails, because record length will be changed. It should be possible to use the CONVERT utility and FDL on the VAX, but there are no experiences.

d1) Transfer from VAX to IBM/MVS

It requires proper structuring of the IBM/MVS file.

You have to type either

```
 quote site recfm=f blocksize=### lrecl=###  ,    on VAX or

 LOCSITE RECFM=F BLOCKSIZE=### LRECL=###  ,      on IBM/MVS.
```

Use of INTERLINK:

```
 COPY  vax_file_name  DSYIBM::"[userid]filename/BINARY"
```

d2) Transfer from VAX to Apollo

It can be triggered on both sides and causes no problems. No declaration of any special file structure is necessary.

d3) Transfer from VAX to Macintosh IIci

It can be triggered on both sides and causes no problems. No declaration of any special file structure is necessary.

B.7.2 What is FFTP and how to use it

'fftp' is an interface to FTP intended to simplify the Apollo user's work while transmission of binary FPACK files from Apollos to the IBM/MVS. It can be used by anybody who has installed the .netrc file in his $HOME directory. This file should contain the following three entries, which can also be put into one line

```
machine mvs
login your_ibm_userid
password your_ibm_passwd
```

Don't worry about making your password public. For the .netrc file you have to specify access permissions to 600 or 660 which means that nobody but you (and the super-user) are allowed to read it. Otherwise FTP will not use your .netrc file and 'fftp' crashes. To set the access mode, just execute the command

```
chmod 600 .netrc
```

after the file is created. .netrc will simplify all your FTP sessions, not only those devoted to binary FPACK files! For details type man FTP on Apollo.
To transmit a binary FPACK file from Apollo to IBM/MVS just type 'fftp', enter the names of the local and remote files and answer one question, when you are prompted. That's all!
(If you are not h1 member, produce a soft link to /h1/bin/fftp from your $HOME/bin directory.)

The rest is done by 'fftp'. It
- checks, whether the local file exists (if not,'open failed for local file' appears on the screen) and whether it is a binary FPACK file, finds the physical record length for the local file and prints the message on the screen,
- connects you to the IBM/MVS file system specified in the .netrc file, according to your decision, deletes or not the remote file if it already existed (This is an important step. If the file already exists, it preserves its RECFM and BLOCKSIZE. These parameters cannot be changed by FTP commands! On the other hand, very large files must be preallocated and, in this case, they must not be deleted by 'fftp'.),
- sets the transfer mode to BINARY,
- sends RECFM and BLOCKSIZE to the IBM/MVS,
- transmits data,
- reports status,

- closes the session, or starts another transfer, according to your
  choice.

B.7.3 NFS transfer of binary FPACK files

This works at present only between Apollo and IBM/MVS. In order to safely exchange files in all possible combinations one has to specify RECFM=F as well as LRECL and BLOCKSIZE during mounting the IBM/MVS file system. This produces a channel for exchanging those files only, which use the LRECL specified during the mounting operation. This is clearly a limitation which can slightly be relaxed by mounting the same IBM/MVS file system several times in different mount points (Apollo pathnames) and with different specifications of LRECL.

```
Examples of possible problems:
------------------------------
- If the LRECL specified during the mount operation does not correspond
  to the actual physical record length, then :
    - files written on Apollo cannot be read neither on IBM/MVS nor even
      on Apollo
    - files written on IBM/MVS cannot be read on Apollo.
- Files written on IBM/MVS with RECFM=VS (e.g. dynamically allocated
  during an interactive session) cannot be OPENed in Fortran programs
  running on Apollo.
```

B.7.4 Remote file support in FPACK

FPACK allows for remote access to all types of binary FPACK files residing on machines on which FPSERVER is being run. There exist versions of FPSERVER for IBM/MVS, for UNIX machines, and for Macintosh. Currently, at DESY, the IBM/MVS, Apollo, and SGI files can be accessed remotely from all platforms. For remote files the user should set the HOST parameter when he opens the file. The system uses RPC and is based on the TCP transmission protocol. FPSERVER is a permanent server which starts private servers under the account of the user who requests a service.

FPACK internally keeps track on connections opened to remote hosts and files opened on these hosts. When a new file is to be opened, FPACK opens a new connection (starts a new private server) only if there is no connection previously opened. Otherwise,the old connection is reused. The remote private server is being started automatically,when necessary, and stopped when the last file on the remote host is being closed. On the other hand, any private server dies if it remains idle for TIMEOUT_S seconds (currently 18000).

If the number of connected hosts exceedes MAXSERV (currently 10), then each next connection to a new host is specific for the file being opened,cannot be reused and is being closed together with the file. The corresponding private server is being stopped at the same time.

On UNIX machines the private server is started in the HOME directory of the user. The FILE parameter should then be given as a full pathname, a relative pathname starting with a slash or a pathname relative to the HOME directory. For MVS files the full data set name should be specified (without apostrophs, however!). On UNIX machines the private server writes messages to the file $(HOME)/fpnet.log. On IBM/MVS there is normally no output from the private server unless the user switches the MSGCLASS to T by setting this option in the HOST parameter (just add $T to the host name and include the whole string in double quotes).

The user should supply his userid and password for the remote machine. This can be done through the file name, e.g.

```
   HOST=DSYIBM  FILE="F99ABC.COSMIC.RUN7@your_userid@your_passwd"
or
   HOST="DSYIBM$T"  FILE="F99ABC.COSMIC.RUN7@your_userid@your_passwd"
or
   HOST=APOLLO FILE="/h1/h1data/COSMIC.RUN7@your_userid@your_passwd"
```

A better and recommended way for sending authentification data is, however, using the so-called "netrc" file on the local host. On UNIX machines it should be installed as .netrc in the HOME directory and the access mode should be set to 600, on MVS this should be your_userid.PER.NETRC, on IBM VM "DOT NETRC *",
and on MacII the file netrc should be installed in the main volume. Don't forget to set the password protection for FTP on your Macintosh before installing netrc. The format of this file is described in section B.7.2. The FILE parameter will then contain the filename only. For previous examples, it would take the form:

```
   HOST=DSYIBM  FILE=F99ABC.COSMIC.RUN7
or
   HOST="DSYIBM$T"  FILE=F99ABC.COSMIC.RUN7
or
   HOST=APOLLO FILE="/h1/h1data/COSMIC.RUN7"
```

The corresponding userid and password will be read from the "netrc" file and automatically sent to the remote host. Note, that quotation marks are needed to prevent FPACK from special treatment of some characters (like /,@,...) and from automatic conversion to upper case characters. Note also that the symbolic names of the hosts used in the examples are specific for DESY. Outside users should consult local network experts on which symbolic names are known to the local naming resolver. It is also possible to set the IP-address to the host parameter. For DSYIBM this should look like HOST="131.169.93.32".

Important remark

If you write a network application which is to run on IBM or IBM/MVS DO NOT FORGET the following:
In order to avoid problems with the communication between Fortran and C on IBM and IBM/MVS, always CALL CDUMMY without arguments at the very beginning of the main program of your Fortran application based on FPACK. This initializes the C-environment for the whole time of execution of your application.

B.8 SHIFT interface

If the F-package library is created with an additional selection SHIFT, the Remote File Access System (RFAS) is used for accessing files stored on SHIFT disk servers. This is the most efficient way of accessing remote files within the SHIFT cluster. In addition, SHIFT tools for reading and writing tapes on the default tape server (IBM mainframe in the DESY DICE cluster) are used. Disk files are declared as SHIFT disk pool files by an additional keyword SHIFT in the F-pack OPEN statement, e.g.

```
OPEN BOSINPUT UNIT=1 FILE="TESTFILE" SHIFT
```

will open an existing file TESTFILE in the user's default pool rather then in the current working directory,

```
OPEN BOSOUTPUT UNIT=2 FILE="TESTFILE" SHIFT NEW RECL=27720
```

will allocate the default space (250MB in the current DICE instalation) in the default user's pool and open the new file TESTFILE there. The NREC parameter in the OPEN statement can be used to change the default maximum file size.

If data from a non-default pool should be accessed, this has to be specified in the file name as "[pool.user]filename", e.g.

```
OPEN BOSINPUT UNIT=1 SHIFT FILE="[user_id]TESTFILE"
```

will open the file TESTFILE in the default pool of the user 'user_id',

```
OPEN BOSINPUT UNIT=1 SHIFT FILE="[pool_name.user_id]TESTFILE"
```

will open the file TESTFILE in the directory of the user 'user_id' in the pool 'pool_name',

```
OPEN BOSINPUT UNIT=1 SHIFT FILE="[pool_name.]TESTFILE"
```

will open the file TESTFILE in the user's partition of the pool 'pool_name'. The period in the last example is obligatory!

If STAGE parameter is used in the OPEN statement within the SHIFT cluster, then file name refers to the file name on the default tape server (IBM mainframe in case of the DESY DICE instalation). No SHIFT keyword is needed in that case. For OLD files, the tape file from the default tape server will be copied onto a local disk (more precisely, into the SHIFT disk pool) with the same file name. For NEW files, the file will be created in a local disk pool first, copied onto a tape on the default tape server upon explicit close and deleted from the local disk pool after successful transmission. For example:

```
OPEN BOSINPUT UNIT=1 STAGE FILE="HERA03.H1DST5.C9200200"
```

will allocate a file HERA03.H1DST5.C9200200 in the user's pool, copy the IBM cartridge HERA03.H1DST5.C9200200 into it and access the local copy afterwards. Multiple input files are allowed. Note, that the generic name on DICE coincides with the entry in the IBM catalogue, including the USERID.

```
OPEN BOSOUTPUT UNIT=2 FILE="ABCXYZ.TESTFILE" NEW &
STAGE RECL=27720
```

will create a file ABCXYZ.TESTFILE in the default pool and result in copyig it to an IBM cartridge ABCXYZ.TESTFILE as soon as the user has closed it explicitly, e.g. by CLOSE BOSOUTPUT. The file will not be sent if it has not been closed properly. SPLITMB can also be used with NEW and STAGE.

In addition, if the keyword SHIFT is not used in the OPEN statement but the filename has the form

```
host:unix_path_name
```

then an UltraNet connection is opened to a deamon running on the 'host' and the file 'unix_path_name' on the 'host' is accessed.

Additional libraries needed for this system to work can be linked through the following entries in the makefile:

```
on SGI
  -L /usr/local/lib -lshift -l rpclib
```

```
  -L /usr/lib -lulsock

on Sun
  -L /usr/local/lib -lshift
  -L /usr/lib -lulsock
```

B.9 FATMEN interface

```
!!!!!   FATMEN is implemented on H1-SGI, HP and FATMEN mechines   !!!!!


FATMEN is a package that provides access to data in a location,  medium
and operating system transparent manner. It is implemented using a file
catalogue and a set of it cooperating servers. It is designed  around a
model approximating to the IEEE Computer Society  Reference  Model  for
Mass Storage Systems.  Data  is  referenced  using  UNIX-style  generic
names.

FPACK has interface to the FATMEN File Catalogue. User  has  access  to
this catalogue by using GENER="..." parameter in FPACK OPEN  statement.

Examples :

For input
OPEN BOSINPUT UNIT=1 GENER="//desy/h1/test.generic.name"
This  command  will  open  the  connection  to   the   FATMEN   machine
(fatmen.desy.de alias rws03.desy.de), will get  the  information  about
file_name, host_name and media_type from the FATMEN catalogue. Then the
file will be opened.

For output
OPEN BOSOUTPUT UNIT=2 GENER="//desy/h1/test.generic.name" &
                      FILE="F99ABC.TEST.FILE.NAME" &
                      RECL=23400 WRITE
This command will open the file. The real entry will  be  written  into
the FATMEN catalogue during the FPACK CLOSE command.
(CALL FPARM('CLOSE ALL') or CALL FPARM('CLOSE BOSOUTPUT'))



The string parameter GENER consists of sequence of generalized  generic
names (ggn) separated by commas in double quotes:

   GENER="ggn1,ggn2,..."

Each ggn has to contain a generic_name, which  is  UNIX  like  symbolic
```

name for the file. Optionaly it can contain selection conditions.

```
ggn ::= [{...}] generic_name [{...}]
```

Generic_name may contain the wildcard characters (*,%,...), so it describes a set of entries in the FATMEN catalogue. The selections conditions are optional and serve to limit the number of possible candidates (entries in catalogue). If the selections condition is located after generic name, it is treated as local one and acts to this generic name only. If the selections condition is located before generic name, it is treated as a global one and acts also to all following generic names in list until this will not be cancelled by another global conditions. The local condition always suppresses the action of global one.

Selection conditions consist of items separated by semicolons (;). Each item contains a keyword and numerical parameters separated by commas or blanks. Using of the format n1-n2 to set the range of numbers is also allowed:

```
   keyword n1[-n2][,]n3[-n4] ...; keyword2 ...
where
   n1,n2,... - positive integer numbers
```

Allowed keywords are: LOCATION, COPYLEVEL, MEDIATYPE, USERWORDS, LOGLEVEL. List of items enclosed in parentheses is the row of selection matrix. Separating of rows by ';' is optional, i.e. right parenthesis ')' is treated as a couple of characters ');'. Matrix defines the order in which selections conditions will be tested. Matrix elements may be only the items with the following keywords: LOCATION, COPYLEVEL, MEDIATYPE. If some keyword is not defined in row the default value -1 is taken that means "no check on this parameter will be made".

Example of selection condition:

```
log 3; u 0 1-15 -1 18; (loc 1-2; m 2,1)(loc 13; c 0-2)
```

will be generated the matrix:

copylevel	mediatype	location
-1	2	1
-1	2	2
-1	1	1
-1	1	2
0	-1	13
1	-1	13
2	-1	13

Set loglevel to 3 (verbose) and userwords as shown.

Appendix A

The F-record structure

```
All records of a file have the same
length (as given as the parameter LIST(1) in the FUPARM call).


1) Record header
----------------

     word   content
        1   NRECWD   number of words in record (including this word)
        2   NUSEDW   number of used words, following

Remarks:
Physical records have a fixed length of NRECWD words (content of  first
word), unless the access is keyed.
For keyed access data sets the length of physical records is  variable,
but not longer than NRECWD. Thus in all cases the maximum record length
can be obtained from any record (all records of a  data  set  have  the
same content in the first word.
NUSEDW is the number of used words in  the  record,  thus  the  maximum
value of this word is NRECWD.


2) Record segment header
------------------------

     word   content
        1   pattern to recognize byte swapping (4,3,2,1)
        2   format code / origin code
        3   record number (*100 + position)                         bytes
        4   name1 (hollerith) (chars 1...4)              key 1    1....
        5   name2 (hollerith) (chars 5...8)              key 2
        6   number1                                      key 3
        7   number2                                      key 4
        8   physical record number within                key 5    ...20
                              logical record
        9   dataword
```

```
       10   segment code (0..3)
       11   number of words following for this segment in this record
```

Remarks: The segment header is always completely in one record (no splitting over record boundaries). There is always a segment header after the record header in words 1 and 2 of a record.

Word 1: The pattern to recognize byte swapping is the integer

```
            1+256*(2+256*(3+256*4)) = 67305985
```

Word 2: The format code / origin code is the integer

```
        (format code * 16 + origin code), where:
format code:    1   IEEE
                2   IBM main frame
                3   VAX
                4   DEC
                ...
origin code:    0 = unknown, 1 = BOS, 2 = LOOK, 3 = BOS & LOOK
```

Word 3: This word contains the number (starting at 1) of the record time 100 plus the number of the segment within the record (counting starts at 0). This record number is stored in directories. There may be up to 100 segments in one record (highest segment number is 99).

Words 4...8: these consecutive words form the record key. The record key as seen by a user has the following parts:

```
  name     (string of up to 8 characters), for example : event name
  number 1 (integer)                                     run number
  number 2 (integer)                                     event number
  number 3 (integer)                                     counter
```

Within records the name is stored in two integers (hollerith). Since long logical records may require several physical records, there is an additional word (number 3) in records, counting the physical records for one logical record (starting at 0). This extra word is necessary for keyed access.

Word 9: this is an extra integer data word, stored together with the key as directory information, but it is not part of the key. This may contain information on classes for certain keys.

Word 10: The segment code (0..3) is defined as follows:

```
                0   complete segment
                1   first segment
                2   middle segment
                3   last segment
```

Word 11: this is the number of words following in the segment in the record.

3) Data segment header

```
word   content
   1   number of words incl format words
   2   name1 (hollerith) (chars 1...4)
   3   name2 (hollerith) (chars 5...8)
   4   number of bank
   5   number of columns (NCOL) (optional)
   6   number of rows    (NROW) (optional)
   7   data segment code (0,1,2,3)
   8   number of data words in previous subsegments
   9   number of data words in this subsegment
  10     1    format (hollerith) (chars 1...4)
 ...     2    ...
 ...          ...
```

Remarks: The data segment header is always completely in one record (no splitting over record boundaries). There is always a data segment header after a record segment header.

Word 1: the number of data words of the data segment header includes the (variable) number of worsds, containing the format.

Words 2 to 4: these words contain the identifier of the data.

Words 5 to 6: these words are optional and contain zero if they are undefined. Usually however at least NCOL will be nonzero. This convention has been introduced to allow very long data parts, which can be stored (written) before the total length is known.

Word 7: the data segment code has the same meaning as the record segment code (see above).

Words 8 to 9: In addition to the number (word 9) of data words in the record there is the number of data words in previous records. This information is necessary to allow a word format conversion of a single record without having information from previous records.

Words 10 and following: the number of data words containing format code is = word(1)-9.

4) Data

```
word   content
   1   first data word of this subsegment
```

```
  2   second data word of this subsegment
...
```

Appendix B

Internal commons

The status information on all F-file/units, kept in a common-array, is described below.

```
    INFU(I,.)
     open parameter
  1  record length in bytes for output
  2  access: 1=seq    2=dir   3=keyed   4=special   5=ordered
  3  form:   1=binary 2=text
  4  action: 1=read   2=write 3=readwwrite  4=modify
  5  word format for output: 0=local 1=IEEE 2=IBM 3=VAX 4=DEC
  6  FORTRAN unit number for the data file or index file
  7  = 0 for NO index file
     > 0 FPACK unit number of the data file ( over index file )
     = -1 data file exist but is not opened yet
  8  UNIX file descriptor or STREAM id for remote files
     if -1 then scratch file opened from FORTRAN
  9  = 0 for local files; = CLIENT id for remote files
 10  = 0 for no index file
     > 0 pointer to INFU of the connected index file

 11  1=RECSEP  logical record separation (for sequential files)
 12  1=NOOPEN  2=DEFER  3=STAGE  5=UNIXTAPE
 13  1=NOCOMP  switch off the automatic data compression on cart.
 14  physical record number after the last SUCCESSFUL I/O
 15  medium for I/O in the C language:  0=disk 1=unixtape 2=rfio
 16  status: 1=old   2=new   3=scratch
 17  = NREC = number of primary records
 18  = NREC2 = number of secondary records
 19  = BLFACTOR = blocking factor
 20  = BUFNO = number of buffers

 21  SPLITMB - split before next logical record ( for output )
 22  1 = SPLITMB reached
 23  number of first file extention.
```

```
         e.g. if the writing started from .B02, then INFU(23,..)=102
    24   1 = suppress data block selections (for index files)
    25   internal flag for ordered access
         0 = normal reading/writing
         1 = directory/map reading writing
         2 = directory/map descriptor reading/writing
    26   internal pointer for ordered access
         current position in directory, when normal data record R/W
         or current position in directory/map descriptor, when
         directory/map R/W (dependent on INFU(25,..))
    27   current physical record number for R/W
    28   current physical record number for reading from
           the connected data file, if it's not opened yet
    29   first possible free record for ordered files
    30   ordered access flag
         0 - non-ordered file
         1 - ordered file opened in sequential mode
         2 - ordered file opened in direct mode

    31   "private cartridge" flag (only IBMDESY)
         0 = usual cartridges from catalogue and available by ACS
         1 = user cartridges (not catalogued, VOLSER=volume_name)
    32   "skip corrupted (incomplete) logical records" flag.
    33   1 = parameter GENER="list_of_generic_names" is given
    34   used for index file only
         = record length of data file on cartridge if it's not
           opened yet. It's used only if index file refers to
           the data files on cartridges.
         = 0 if the data file is on disk (direct access is used)
    35   = DUNIT = FORTRAN unit number of data file on the cartridge,
           (over index file). DUNIT for disk files is optional.


         general status
ISGEN+ 1   0=undef 1=reading 2=eofreading 3=writing 4=eofwr 5=rewind
       2   = origin code (set by FWPARM('ORIGIN',VALUE)
       3   = nr of columns/item in text output (FWPARM('TEXTCOL',VALUE)
       4   = 0 convert (normal) = 1 inhibit conversion on output
       5
       6   flag for lock/unlock
       7   flag for lock/unlock
       8
       9
      10


         read-status
IRSTU+ 1   -1 record not converted  +1 record converted
       2   error code of last read operation
```

```
        3   error code of current logical record (0=no error)
        4   =1 if key just read
        5   record number after last read operation (except keyedacc)
        6   error code
        7   pointer to record segment header in connected data
               file, if it is not opened yet
        8   IP = pointer to record segment header
        9   JP = pointer to data segment header
       10   KP = pointer to data

       11   =0 for read, =1 for disturbing write   (keyed access)
       12   IDIR (=0    ,+-1 forward/backward last, +-2 requsted)
       13   =1 for rewind
       14   name1         last   requested record key
       15   name2         -"-      -"-
       16   nr1           -"-      -"-
       17   nr2           -"-      -"-
       18   order         -"-      -"-
       19   =0 first part, =1 not first
       20   format code / origin code after swapping before conversions


            read-statistics
  IRSTA+ 1
        2   Number of physical records read (from FRREC)
        3   Number of logical records read (after RSELECT, from FRKEY)
        4   (Number of bytes read)/1000  ( from FRREC )
        5   Number of data blocks read (from FRHDR)
        6   Number of logical records read (before RSELECT, from FRKEY)
        7   MOD(Number of bytes read,1000)  (from FRREC)
        8
        9
       10


            record/segment header (read)
  IRKEY+ 1  byte order control word
        2   format code / origin code
        3   record number (*100 + position)
        4   name1
        5   name2
        6   nr1
        7   nr2
        8   order/cycle/version
        9   data word
       10   segment code (0 1 2 3)

       11   NWORDS (of segment following)


            data header (read)
```

```
IRHDR+ 1   number of words
       2   name1
       3   name2
       4   number
       5   ncol
       6   nrow
       7   data segment code
       8   number of data words in previous segments
       9   number of data words in current segment
      10   fmt1
     ...   fmt2
           fmt3
           ......
```

```
           write-status
IWSTU+ 1   =0 no buffer;   =1 buffer
       2   format code/origin code
       3   record number after last write operation (except keyed acc)
             in case of direct access write it is allowed to set it by
             new record number
       4   key info: 0 = undef, 1 = def, 2 = in rec
       5   hdr info: 0 = undef, 1 = def
       6   error code
       7   position of logical record within physical record
       8   IP = pointer to record segment header
       9   JP = pointer to data segment header
      10   KP = pointer to data

      11   =1 if class bits selected    directory info
      12   mask for class word             -"-
      13   =1 if reset of class word       -"-
      14
      15
      16
      17
      18
      19
      20
```

```
           write-statistics
IWSTA+ 1   Physical record counter
       2   Number of physical records written ( from FWREC )
       3   Number of logical records written ( from FWKEY )
       4   (Number of bytes written)/1000 ( from FWREC )
       5   Number of data blocks written ( from FWHDR )
       6   Number of data blocks written without format ( B32 added )
```

```
                                                         ( from FWHDR )
        7   MOD(Number of bytes written,1000) ( from FWREC )
        8
        9
       10


            record/segment header (write)
 IWKEY+ 1   byte order control word
        2   format code / origin code
        3   record number (*100 + position)
        4   name1
        5   name2
        6   nr1
        7   nr2
        8   order/cycle/version
        9   data word
       10   segment code (0 1 2 3)

       11   NWORDS (of segment following)


            data header (write)
 IWHDR+ 1   number of words
        2   name1
        3   name2
        4   number
        5   ncol
        6   nrow
        7   data segment code
        8   number of data words in previous segments
        9   number of data words in current segment
       10   fmt1
      ...   fmt2
            fmt3
            ......
```

Appendix C

Internal IOPEN array

```
 I   IOPEN(I)

 1   unit number (UNIT)
 2   physical record length (RECL)
 3   number of records - primary space (NREC)
 4   number of records - secondary space (NREC2)
 5   blocking factor (BLFACTOR)

 6   number of buffers (BUFNO)
 7   not yet used
 8   number of MB per file (SPLITMB)
 9   action  ( 1=read   2=write  3=readwrite  4=modify )
10   status  ( 1=old    2=new    3=scratch )

11   access  ( 1=seq    2=dir    3=keyed   4=special  5=ordered )
12   form    ( 1=binary 2=text )
13   wordfmt ( 0=local  1=ieee   2=ibm     3=vax      4=dec )
14   recsep  ( 1=recsep, logical records separating )
15   noopen  ( 1=noopen, open should NOT be done )
             ( 2=defer, DD card with DEFER parameter exists )
             ( 3=stage )   (4 = stagedeb = stage with debug )
             ( 5=unixtape = tapes on UNIX machines )

16   compact ( 1=compact=default  2=nocompact for export )
17   catalog ( 1=shift )
18   skipcorev ( 1=skip corrupted event at the end of file )
19   data file (son) unit number (DUNIT)
20   flag "FATMEN catalog used" (1=used)
```

Appendix D

Example of output in text mode

```
RECL = 80 ! RECL= record length in bytes
RECORD   CJCPATT 4711                              ! start of F-record
FIGURE   13   FMT=(I)                                       ! FIGURE
TEXT     3   FMT=(A)                                        ! TEXT
'Vertex distribution '
HS       1   FMT=(A)                                        ! HS
'HIST DISC WHITE     '
BINS     0   FMT=2(I,2F)                                    ! BINS
100 0. 400.
STAT     0   FMT=I,10F                                      ! STAT
1020 1020. 1020. 0.12851 299.802 147.711 87.7473
ENTRIES  0   FMT=(I)                                        ! ENTRIES
0 1020 0
CONTENT  1 100   FMT=(F)                                    ! CONTENT
21. 15. 12. 17. 7. 13. 14. 13. 8. 20. 14. 20. 19. 15. 9. 15. 19. 14.
8. 8. 18. 14. 10. 15. 15. 12. 19. 13. 16. 10. 8. 18. 22. 16. 16. 10.
10. 11. 10. 11. 7. 20. 12. 11. 11. 13. 12. 15. 7. 15. 18. 19. 7. 17.
16. 7. 20. 10. 12. 14. 13. 8. 11. 18. 12. 19. 14. 14. 11. 14. 14. 10.
16. 12. 16. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0.
FIGURE   70   FMT=(I)                                       ! FIGURE
TEXT     3   FMT=(A)                                        ! TEXT
'This is text under number three '
HD       1   FMT=(A)                                        ! HD
'LEGO     WHITE     '
BINS     0   FMT=2(I,2F)                                    ! BINS
50 0. 300. 20 0. 0.01
STAT     0   FMT=I,10F                                      ! STAT
1020 1020. 1020. 0.123525 299.793 147.706 87.7471 5.2099E-6
0.00997976 0.00499357 0.00292555
ENTRIES  0   FMT=(I)                                        ! ENTRIES
0 0 0 0 1020 0 0 0 0
CONTENT  1 50   FMT=(F)                                     ! CONTENT
2. 1. 3. 1. 0. 1. 1. 2. 0. 2. 1. 3. 0. 1. 1. 0. 1. 1. 0. 0. 2. 1. 1.
0. 1. 0. 1. 2. 1. 1. 0. 1. 1. 0. 1. 2. 1. 0. 1. 2. 0. 1. 2. 1. 0. 1.
4. 0. 1. 2. 1. 0. 0. 0. 1. 1. 1. 1. 0. 0. 0. 0. 1. 0. 2. 3. 1. 0. 2.
```

```
2. 1. 2. 2. 1. 4. 2. 0. 1. 0. 2. 2. 1. 2. 1. 1. 0. 3. 3. 1. 2. 0. 1.
0. 2. 1. 0. 0. 1. 2. 1. 1. 0. 2. 0. 1. 1. 2. 1. 3. 0. 3. 1. 0. 0. 0.
0. 3. 1. 1. 0. 0. 1. 1. 3. 1. 1. 0. 1. 1. 2. 0. 2. 1. 1. 1. 1. 2. 1.
0. 1. 1. 2. 3. 0. 1. 1. 1. 1. 2. 1. 1. 1. 3. 3. 1. 0. 2. 1. 0. 3. 4.
0. 1. 0. 1. 2. 3. 0. 0. 1. 0. 2. 1. 0. 0. 0. 1. 2. 1. 0. 0. 1. 2. 2.
1. 0. 1. 1. 3. 0. 1. 1. 1. 1. 1. 1. 1. 2. 2. 2. 3. 2. 1. 0. 1. 1. 0.
1. 1. 0. 1. 4. 0. 1. 1. 0. 2. 2. 0. 2. 0. 3. 3. 0. 0. 0. 0. 0. 0. 1.
1. 1. 1. 0. 0. 1. 1. 0. 0. 4. 0. 2. 1. 0. 0. 3. 0. 0. 1. 0. 0. 0. 1.
0. 2. 0. 2. 2. 0. 2. 1. 0. 0. 1. 0. 0. 2. 0. 3. 1. 2. 0. 2. 3. 1. 1.
1. 1. 2. 0. 3. 0. 0. 1. 0. 1. 0. 1. 2. 0. 1. 1. 1. 2. 1. 1. 0. 1. 0.
1. 0. 1. 3. 1. 0. 1. 0. 1. 2. 1. 1. 0. 0. 2. 2. 0. 1. 6. 1. 1. 1. 0.
1. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1. 2. 1. 3. 1. 1. 1. 0. 2. 0. 2. 2. 0.
0. 0. 0. 2. 2. 2. 0. 1. 2. 1. 0. 2. 1. 2. 1. 1. 0. 0. 1. 0. 1. 1. 1.
1. 2. 0. 2. 1. 0. 0. 1. 1. 1. 1. 1. 2. 0. 1. 1. 1. 3. 1. 1. 1. 1. 2.
1. 0. 1. 2. 0. 1. 0. 0. 2. 0. 2. 3. 1. 1. 1. 2. 0. 1. 2. 2. 2. 3. 1.
1. 3. 0. 1. 1. 1. 2. 0. 1. 0. 1. 1. 0. 2. 1. 1. 0. 0. 0. 3. 0. 1. 0.
0. 0. 0. 1. 1. 1. 1. 1. 0. 1. 2. 0. 0. 2. 0. 0. 0. 2. 1. 0. 2. 2. 1.
1. 2. 0. 1. 1. 0. 1. 2. 1. 1. 0. 3. 0. 0. 0. 0. 0. 1. 3. 1. 0. 2. 0.
0. 2. 0. 2. 1. 1. 0. 0. 0. 3. 1. 3. 1. 0. 0. 1. 0. 2. 3. 1. 1. 2. 3.
0. 3. 2. 0. 1. 4. 0. 0. 1. 0. 0. 1. 2. 1. 0. 1. 1. 1. 2. 0. 1. 0. 0.
0. 0. 2. 0. 0. 3. 0. 1. 1. 1. 0. 1. 0. 1. 1. 2. 2. 2. 1. 2. 0. 4. 1.
1. 0. 0. 0. 2. 2. 1. 0. 1. 0. 0. 0. 1. 1. 0. 2. 0. 0. 1. 0. 2. 2. 2.
2. 2. 0. 1. 1. 2. 0. 0. 0. 1. 2. 2. 1. 0. 0. 1. 1. 1. 2. 2. 2. 2. 1.
1. 3. 3. 0. 1. 1. 3. 0. 2. 1. 0. 1. 0. 1. 0. 0. 2. 0. 0. 0. 2. 1. 1.
2. 2. 1. 1. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 2. 0. 2. 0. 1. 0. 1.
0. 0. 2. 2. 1. 0. 1. 0. 0. 0. 0. 1. 0. 2. 2. 0. 0. 0. 2. 1. 0. 2. 0.
0. 1. 0. 0. 1. 2. 0. 1. 0. 2. 3. 0. 1. 2. 3. 1. 0. 2. 0. 0. 1. 2. 2.
2. 0. 2. 2. 1. 1. 2. 1. 0. 2. 1. 2. 1. 0. 1. 1. 2. 3. 3. 1. 2. 1. 1.
0. 1. 1. 0. 1. 0. 0. 0. 3. 2. 1. 2. 1. 0. 2. 1. 0. 0. 0. 2. 2. 0. 1.
1. 2. 1. 1. 2. 2. 3. 1. 2. 0. 1. 1. 1. 3. 3. 0. 0. 0. 0. 2. 3. 1. 0.
2. 0. 0. 4. 1. 0. 1. 2. 1. 2. 0. 1. 3. 0. 0. 1. 2. 1. 0. 1. 1. 2. 2.
1. 3. 1. 0. 1. 2. 0. 0. 2. 0. 1. 1. 1. 1. 1. 2. 1. 1. 1. 1. 1. 1. 1.
0. 1. 1. 3. 0. 1. 1. 0. 3. 0. 2. 0. 2. 2. 2. 1. 1. 1. 0. 0. 0. 2. 0.
1. 1. 1. 1. 0. 0. 2. 2. 0. 2. 0. 3. 0. 1. 0. 1. 1. 1. 0. 1. 1. 2. 1.
2. 0. 2. 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 2. 3. 3. 2. 1. 1. 3. 2. 1. 1.
2. 0. 0. 3. 1. 0. 1. 2. 5. 3. 2. 0. 0. 1. 1. 0. 0. 0. 0. 2. 1. 0. 1.
0. 2. 1. 1. 1. 0. 0. 1. 1. 2. 0. 2. 1. 3. 1. 1. 2. 2. 1. 1. 0. 2. 0.
1. 4. 1. 1. 0. 0. 1. 0. 1. 2. 1. 1. 1. 0. 1. 1. 1. 2. 0. 1. 0. 1. 1.
1. 2. 2. 0. 0. 0. 1. 2. 0. 0. 2. 1. 0. 2. 1. 1. 0. 1. 2. 0. 5. 0. 1.
2. 0. 1. 0. 0. 1. 2. 0. 0. 1. 2. 2. 2. 0. 1. 1. 0. 2. 2. 2. 1. 1. 0.
2. 0. 0. 1. 2. 0. 0. 2. 1. 0. 0.
END-OF-RECORD
```

Appendix E

File attributes and open statements

```
The routine FUPARM is called from the routines FPACK/FPARMR. It is  not
necessary for users to call FUPARM in their main program. It should  be
used in very special cases only.

Before any other calls, the user has to specify the characteristics  of
IO units to the FPACK system, by a call to the subroutine FUPARM, in an
array LIST with certain numerical  parameters;  see  Table  3  for  all
details. In addition the user  may  specify  the  symbolic  file  name,
called DATANAME,  for  the  file.  The  user  may  either  specify  the
(FORTRAN) unit number of the file, or he may ask  FUPARM  to  return  a
"free" unit number.

In the FPACK calls files can be either specified by  a  numerical  unit
number or by a symbolic file name, the so-called data name, which  is a
string of up to 16 characters.  Specification  of  the  numerical  unit
number, the data name (optional) and other characterics is  done  by  a
call to FUPARM.


+----------------------------------------------------------------------+
|  INTEGER LIST(6)                                                     |
|              --- -------- ----                                       |
|  CALL FUPARM(LUN,DATANAME,LIST)                                      |
|                                                                      |
|  Arguments (all are input arguments)                                 |
|     LUN      = unit number (integer) *)                              |
|     DATANAME = associated symbolic file name (up to 16 characters)  |
|     LIST     = array of numerical parameters                         |
|      LIST(1) = record length in bytes for output (multiple of 360)  |
|      LIST(2) = access: 1=seq    2=dir   3=keyed   4=special          |
|      LIST(3) = form:   1=binary 2=text                               |
|      LIST(4) = action: [1=read   2=write 3=readwrite]                |
|      LIST(5) = wordformat 0 = local 1=IEEE 2=IBM 3=VAX for output   |
|      LIST(6) = 0 data files, = data file unit for directory file**) |
|                                                                      |
|  *)  If  LUN = 0 at call, a free unit number is returned.            |
```

```
|      If  LUN < 0 at call indicates a unit NOT considered to be free |
|                (other arguments meaningless in this case)          |
|                                                                    |
|  **) If LIST(6) = 0 then the file is a data file                   |
|      If LIST(6) not zero, the the file is a directory file          |
|                                                                    |
+--------------------------------------------------------------------+
```

Table 3. Specification of a file attribute to the FPACK.

Note that the word-format for output can be selected by LIST(5). For input files this information is taken from the file.
If LUN=0 at entry to the subroutine, a "free" unit number is returned. This feature can work only if the user marks all units used outside FPACK. All units used outside FPACK have to be marked by the following calls:

```
   CALL FUPARM(-LUN,' ',LIST)   mark unit as reserved
```

Opening of FPACK files with sequential unformatted access

Note that the record length in bytes has to be specified as LIST(1) in the FUPARM call!

IBM-MVS

The record format is fixed, unblocked. The blocksize should be large (and a multiple of 360); the value 23400 seems to be a good choice for 3380 disks and tapes/cassettes.
Example of DD-statement:

```
   RECFM=F,BLKSIZE=23400,LRECL=23400
```

(other record formats would be possible, but will cause errors after NFS or FTP transmission on other computers).

MACII

The option RECORDTYPE='STREAM' has to be used in the OPEN statement.

APOLLO

No special files structure has to be declared.

VAX

The options RECORDTYPE='FIXED',RECL=NBYTES/4 have to be used in the OPEN statement. On the VAX the record length is specified in units of

WORDS instead of bytes, thus the byte-recordlength NBYTES has to be divided by four.

```
Opening of FPACK files with text (sequential formatted) access
---------------------------------------------------------------
```

The record format is fixed blocked (or fixed), the record length can be between 80 and 132 (or 256?). It is recommended to select blocking.
Example of DD-statement:

```
   RECFM=FB,BLKSIZE=9040,LRECL=80
```

```
Examples of OPEN statements:
----------------------------
```

a) On IBM MVS

```
     OPEN(UNIT=1,
    +     FILE=FILEMANE(1:FILENAME_LENGTH),
    +     FORM='UNFORMATTED',
    +     IOSTAT=IOS)
     IF(IOS.NE.0) WRITE(*,*) 'error during opening a file'
```

b) On Apollo
No declaration of any special file structure is necessary. On Apollo FPACK internally performs I/O by calling lower level system routines.

```
     OPEN(UNIT=1,
    +     FILE=FILEMANE(1:FILENAME_LENGTH),
    +     FORM='UNFORMATTED',
    +     IOSTAT=IOS)
     IF(IOS.NE.0) WRITE(*,*) 'error during opening a file'
```

c) On Macintosh IIci
You have to use the option RECORDTYPE='STREAM' in the OPEN statement.

```
     OPEN(UNIT=1,
    +     FILE=FILEMANE(1:FILENAME_LENGTH),
    +     RECORDTYPE='STREAM',
    +     FORM='UNFORMATTED',
    +     IOSTAT=IOS)
     IF(IOS.NE.0) WRITE(*,*) 'error during opening a file'
```

d) On VAX

You have to use the option RECORDTYPE='FIXED' and RECL='physical record length' in the OPEN statement. Very important is to give 'physical record length' in WORDS for binary (unformatted) files and in BYTES for text (formatted) files.

```
      OPEN(UNIT=1,
     +     FILE=FILEMANE(1:FILENAME_LENGTH),
     +     RECORDTYPE='FIXED',
     +     RECL=RECORD_LENGTH_IN_WORDS,
     +     FORM='UNFORMATTED',
     +     STATUS='UNKNOWN',
     +     IOSTAT=IOS)
      IF(IOS.NE.0) WRITE(*,*) 'error during opening a file'
```

The information RECORDTYPE and RECL is not necessary for reading.

e) On SUN and DEC

```
      CALL SOPEN(LUN,DATANAME,ACTION,IOS)

      e.g. CALL SOPEN(5,'INPUT','READ',IOS)
           IF(IOS.NE.0) WRITE(*,*) 'error during opening a file'
```

Appendix F

Utility subprograms

```
               ----
CALL TXTOHL(TEXT,IHL,NHL)        copy text from character string
                     --- ---         into hollerith array

where: TEXT   =  character string of text
       IHL(.) =  integer array, filled with hollerith text, 4/word
       NHL    =  number of words used


            --- ---
CALL HLTOTX(IHL,NHL,TEXT,NCH)    copy hollerith text into character
                        ---- ---     string (reverse to TXTOHL)
where: IHL(.) =  integer array, filled with hollerith text, 4/word
       NHL    =  number of elements in array IHL
       TEXT   =  character string of text
       NCH    =  number of characters
```

Appendix G

Machine-dependent code

```
FORTRAN logical functions
-------------------------


IBM, IBMMVS, VAX, VAXSTN, DECSTN, ALLIANT, MACII
------------------------------------------------
K = IAND(I,J)     logical AND
K = IOR(I,J)      logical OR
K = IEOR(I,J)     logical exclusive OR
K = NOT(I)        logical complement
K = ISHFT(I,M)    right shift for M < 0 ; left shift for M > 0


APOLLO
------
K =  AND(I,J)     logical AND
K =  OR(I,J)      logical OR
K =  EOR(I,J)     logical exclusive OR
K = NOT(I)        logical complement
K = RSHFT(I,M)    right shift
K = LSHFT(I,M)    left shift


SUN
---
K =  AND(I,J)     logical AND
K =  OR(I,J)      logical OR
K =  EOR(I,J)     logical exclusive OR
K = NOT(I)        logical complement
K = RSHIFT(I,M)   right shift
K = LSHIFT(I,M)   left shift
```

Appendix H

Creating of VSAM files

```
VSAM files can be created with special job  only.  First  access  after
creating or erasing of VSAM files must be done with with STATUS=NEW.


Example of a job:
-----------------

//CRVSAM    JOB  ,CLASS=E,TIME=(0,01),NOTIFY=F99ABC
/*JOBPARM LINES=2
//A EXEC PGM=IDCAMS,REGION=512K
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
 DELETE                  /* delete old file, if exists */ -
 (F99ABC.KEYEDACC.FILE) -
 PRG    CLUSTER

 DEFINE CLUSTER          /* create new keyed access file */ -
 (NAME(F99ABC.KEYEDACC.FILE) -
 VOLUMES(FAST15) -
 INDEXED -
 RECORDS(10 10)          /* primary secondary_space (in records) */ -
 KEYS(20 20)             /* length and offset (in bytes) */ -
  SHAREOPTIONS(4 3)      /* multi read + multi write */ -
 SPEED -
 SPANNED -
 REUSE -
 RECORDSIZE(23400 23400))      /* average maximum (in bytes) */ -
 DATA -
 (NAME(F99ABC.KEYEDACC.FILE.DATA)) -
 INDEX -
 (NAME(F99ABC.KEYEDACC.FILE.INDEX))
```

Appendix I

Using of cartridges with DEFER

```
Example of a job:
-----------------

//CRTNEW  JOB  ,CLASS=A,TIME=(0,24),NOTIFY=F99ABC
//*MAIN RELPRI=MED
//*
// EXEC VFORTCLG,
//      CRGN=8000K,CPRM='DC(BCS)',LPRM='AMODE=31,RMODE=24',GRGN=8000K,
//      LLB1='HERA01.H1.FPACK.LOAD',
//      LLB2='HERA01.H1UTIL.LOAD',
//      LLB3='HERA01.H1.BOS.LOAD',
//      LLB4='HERA01.H1.LOOK.LOAD',
//      LLB5='R01UTL.CERN.NEW.PACKLIB'
//C.SYSIN DD *
***********************************************************************
*
*     PROGRAM CRTNEW   -   Allocate a new file on a cartridge
*
***********************************************************************
*
      CALL FPARMR(5)
*
      ...
      ...
      ...
*
  100 CALL FPSTAT
      CALL FPARM('CLOSE ALL')
*
      STOP
      END
//G.SYSIN DD *
//G.FT15F001 DD UNIT=(CART,,DEFER),DISP=
OPEN FPACKIN UNIT=1 READ FILE="F99ABC.TEST.DISC.DATA"
OPEN FPACKOUT UNIT=15 WRITE NEW RECL=23400 DEFER &
```

```
FILE="F99ABC.TEST.CART.DATA"
```

Appendix J

Using of cartridges with STAGE

```
Example of a job:
-----------------

//CRTOLD  JOB  ,CLASS=A,TIME=(0,24),NOTIFY=F99ABC
//*MAIN RELPRI=MED
//*
// EXEC VFORTCLG,
//     CRGN=8000K,CPRM='DC(BCS)',LPRM='AMODE=31,RMODE=24',GRGN=8000K,
//     LLB1='HERA01.H1.FPACK.LOAD',
//     LLB2='HERA01.H1UTIL.LOAD',
//     LLB3='HERA01.H1.BOS.LOAD',
//     LLB4='HERA01.H1.LOOK.LOAD',
//     LLB5='R01UTL.CERN.NEW.PACKLIB'
//C.SYSIN DD *
***********************************************************************
*
*     PROGRAM CRTOLD   -   Allocate an old file on a cartridge
*                          and stage it
*
***********************************************************************
*
      CALL FPARMR(5)
*
      ...
      ...
      ...
*
  100 CALL FPSTAT
      CALL FPARM('CLOSE ALL')
*
      STOP
      END
//G.SYSIN DD *
OPEN FPACKIN UNIT=1 READ STAGE FILE="F99ABC.TEST.DATA.CART"
```

Appendix K

Internal structure of an ordered access file

```
Record structure in an ordered access file is the same as in any
FPACK-file. The diffirence is that in ordered access file logical
record must not occupy adjacent physical records and can be
scattered all over the file.
Ordered access file contains two logical records, with the names
'HEADER' and 'FATDIR', which are hidden from user, and cannot
be read or written by him. FPACK reads, creates and updates
them itself.

First of them ('HEADER') is always placed in one physical record and it
is the first physical record of the file. It contains the only
data bank, called 'HEADER', also.
The contents of the 'HEADER' bank is a sequence of integers, which give
record numbers for physical records, keeping the 'FATDIR'
logical record. The end of sequence is marked by zero.
E.g. if the 'HEADER' bank of the 'HEADER' record contains numbers
124,16,3,0,... then first part of 'FATDIR' record is placed in
physical record 124, the continuation of it in record 16, and the
tail of 'FATDIR' record in record number 3. In some cases 'FATDIR'
record occupies not all of the physical records pointed by 'HEADER'
bank. The last of them may be unused.

'FATDIR' record contains two banks, named 'FAT' and 'DIR'.

'DIR' bank contains repeated combination of words:
   Word 1         first 4 chars of record name, code ASCII
   Word 2         next  4 chars of record name, code ASCII
   Word 3         first number (e.g. run number)
   Word 4         second number (e.g. event number)
   Word 5         classification bits
   Word 6         number of physical record, which contain the
                  first (or only) part of logical record, having
                  the key formed by first 4 words.
```

Each six words contain information on one logical record.
Information on 'HEADER' and 'FATDIR' records is not included in the 'DIR' bank. Information in 'DIR' bank is sorted by record keys.

'FAT' bank contains an array IARR of integers with the meaning:

```
  IARR(I) = -1       I-th physical record of the file is free
  IARR(I) = 0        I-th physical record of the file is the last
                     (or only) physical record of some logical record
  IARR(I) = J, J>0   Physical record J is logically next to physical
                     record I in some logical record.
```

'HEADER' and 'FATDIR' always correspond to each other. If you cancel your job, which were modifying ordered access file, only two cases are possible, regardless of the moment when cancel was done:

1. ALL modifications are lost, file remains the same as before job start. 'HEADER' and 'FATDIR' records are unchanged.
2. ALL modifications are done. 'HEADER' and 'FATDIR' records reflect new file state.

Appendix L

The syntax for the FPACK OPEN statement on the IBM VM

```
1) Cartridges

- OPEN BOSINPUT READ FILE='cartn'       STAGE
  is equivalent to the staging of the cartridge cartn.
  STAGE is supported only for input.

- OPEN BOSINPUT READ FILE='cartn'       DEFER
  is equivalent to SETUP + FILEDEF + MOUNT of the cartridge cartn.
  DEFER can be used for input and output.

- A cartridge is given by CHARACTER*6 XXXXXX with 3 or 4 digits on  the
  right-hand side.

- A range of cartridges can be given by cartn-cartm  for  a  series  of
  cartridges with consecutive numbers.

- A specific file on a multi-file cartridge can be given by
            cartn.m (m=file number)

- A range of files on the same cartridge can be given by
            cartn.i-cartn.j

ATTENTION:  The  2  sorts  of  ranges  can  NOT  be   combined!   (i.e.
cartn.i-cartm.j will be interpreted as range from file i to file  j  on
the same cartridge cartn).

The normal FPACK syntax "cart1,...,cartn" can be used in addition, thus
      "cart1-cart5,cart6.3-cart6.7"
is correct (cart.. representing the syntax as defined above).

2) Files

Concatenation of disk files is not yet implemented.
```

3) Server

An FPSERVER for VM machines is available. It accepts all of the above syntax except that remote setup of cartridges (DEFER parameter) is not allowed.

ATTENTION: as a temporary measure, the syntax for reading a disk file remotely on IBM VM is

FILE="<owner.addr.>filename.filetype" with obligatory quotes,

whereas for local reading the syntax is

FILE="filename filetype"

This will be changed as soon as a correct solution to add a GIME is implemented.

Appendix M

Calling sequences